# Správa linuxového serveru Debian

## Správa linuxového serveru: Úvod

### Linuxový server

Na Linuxu se dá postavit ledacos, od malého LAMP servříku po komplexní řešení firemního informačního systému s vysokou dostupností. V tomto seriálu se zaměřím spíše na onu první oblast, přičemž věnovat se budu nejenom konkrétním postupům, jak nastavit jednoduchý LAMP server, ale i řadě témat spojených se správou serverů, zejména pak bezpečnosti a monitorování.

U čtenářů budu předpokládat základní znalosti prostředí i správy GNU/Linuxu, tzn. správu softwaru, adresářovou strukturu, práci v příkazové řádce, atd. - linuxový začátečník by se neměl pouštět do správy "ostrého" serveru připojeného k Internetu. Vhodnější je začít buď za pomoci virtualizace, nebo přestavbou staršího počítače na server v domácí síti.

### Linuxové distribuce pro server

Při výběru distribuce pro server je třeba se odrazit od požadavků - jaký server chcete postavit (jaký software má mít v repositářích), preferujete-li aktuálnost balíčků před stabilitou nebo naopak (obojí mít nejde - viz dále), ale také třeba jaké máte zkušenosti - jaké distribuce znáte a jste schopni hladce spravovat (nasadit na server distribuci, ve které se nevyznáte, není nejlepší nápad).

#### *Vývojové cykly distribucí*

Jedna z veledůležitých věcí, které je třeba zvážit, je vývojový cyklus vámi zvolené distribuce. Existují v zásadě dva typy vývojových cyklů - periodický a kontinuální (rolling release).

Periodický vývojový cyklus zná asi každý - distribuce udržuje vývojovou větev, kam umisťuje nové balíčky. Jednou za čas se tato větev "zmrazí" (přestane akceptovat nové verze balíčků, akceptuje pouze opravy chyb), otestuje se, odladí se chyby a za nějakou dobu se z této větve vyloupne další vydání distribuce (třeba aktuální Debian 5.0 nebo Ubuntu 9.10). Balíčky v tomto vydání (které se obvykle označuje jako stabilní větev) jsou po určitou dobu udržované - setrvávají sice obvykle ve stejné verzi, ale opravují se u nich zjištěné chyby. Tato doba trvá obvykle do té doby, než se vyloupne další stabilní verze a nějakou dobu se na světě ohřeje (což dává správcům serverů čas naplánovat příslušný upgrade).

Méně známý typ vývojového cyklu mají distribuce jako Gentoo nebo Arch Linux, a sice kontinuální vývoj (rolling release). V rámci něj nedochází k periodickému vydávání nových verzí distribuce, ale k plošné, kontinuální aktualizaci všech balíčků - distribuce má tedy stále aktuální software (bleeding edge). To nevylučuje přítomnost testovacích větví s nejnovějšími balíčky, které se do "stabilní" větve distribuce dostanou až v okamžiku, kdy byly otestovány a bylo shledáno, že fungují.

Kontinuální vývoj zaručuje aktuální software, ale obvykle vede k o něco menší stabilitě (na dostatečné otestování obvykle není čas) a/nebo k problémům při aktualizacích. Dochází totiž ke změnám verzí programů, což obvykle znamená změny v jejich konfiguraci, a pro vás jako správce to znamená, že při každé aktualizaci může program (zejména pak démon) přestat spolupracovat s naším pečlivě odladěným konfiguračním souborem, protože se mezi tím pozměnila jeho syntaxe. Tyto typy distribucí před takovými změnami obvykle varují, což ovšem znamená, že by správce měl před každou aktualizací za domácí úkol projít web distribuce, jestli tam náhodou není napsáno, že po aktualizaci nějaký program (nebo hůř - systémová komponenta) přestane fungovat, pokud se adekvátně neupraví konfigurační soubor.

Naopak periodický vývojový cyklus zajišťuje obvykle o něco větší stabilitu a vzhledem k tomu, že verze programů jsou zmrazeny, nedochází až na výjimky k větším změnám při aktualizacích. Problémem u tohoto typu vývojového cyklu je, že s rostoucím časem software v distribuci zastarává.

#### *Enterprise distribuce a distribuce s dlouhou podporou*

Enterprise distribuce s periodickým vývojovým cyklem jako Red Hat Enterprise Linux (RHEL), SUSE Linux Enteprise Server (SLES) či Mandriva Enterprise Server se vyznačují nejenom komerční podporou, ale i relativně dlouhým obdobím, kdy jsou vydávány aktualizace. U RHEL je to třeba 7 let, i když jisté záležitosti jsou řešeny pouze první čtyři roky (update instalačních médií, apod.) u příslušného vydání.

Dobrou zprávou je, že i některé nekomerční distribuce mají dlouhou podporu, například klon RHEL s názvem CentOS nebo Ubuntu LTS. V obou případech však před případným nasazením doporučuji důkladně projít, které věci se řeší prvních pár let a co se pro zbytek inzerované doby podpory už neřeší (zejména pak u Ubuntu LTS seznamy balíčků, ke kterým už nevychází ani bezpečnostní aktualizace po skončení prvních tří let podpory).

Pro správce serverů jsou takové distribuce z jistého pohledu požehnáním - není třeba řešit upgrade každých 6-12 měsíců jako u většiny ostatních distribucí. Samozřejmě je třeba zvážit, že software v takové distribuci zastarává, a za několik let může být rozdíl mezi softwarem v distribuci a aktuálními verzemi propastný.

#### *Ostatní faktory*

Při výběru distribuce pro server vás bude zajímat ještě pár věcí. Předně to bude počet vývojářů a rychlost reakcí na problémy (u menších distribucí může být v tomto směru problém). A určitě vás bude zajímat i množství softwaru v repositářích - kupříkladu, mně osobně v základu RHEL/CentOS citelně chyběl OpenVPN server. Takový nedostatek sice někdy lze vyřešit přidáním dalšího repositáře s požadovaným softwarem, ale v takovém případě si musíte uvědomit, že závisíte nejenom na distributorovi, ale i správci daného repositáře. Od něj potřebujete nejenom řádné testování, ale i reakce na případné nalezené chyby a bezpečnostní problémy. Bude je řešit podobně jako distribuce (patchováním a nikoliv změnou verze), nebo do repositářů hodí verzi novou, kde se třeba změnila syntaxe konfiguračních souborů?

### Debian GNU/Linux

Debian je druhou nejstarší žijící distribucí vůbec (hned po Slackwaru). Byl založen v roce 1993 Ianem Murdockem a dnes je jednou z nejznámějších, nejrozsáhlejších a nejuznávanějších distribucí vůbec. Debian je ryze nekomerční a se svými uživateli uzavírá "společenskou smlouvu" (Social contract), ve které se zavazuje mj. k tomu, že zůstane 100% svobodný, zaměřený na uživatele a otevřený (v tom smyslu, že před nimi nebude skrývat známé problémy).

Z Debianu vycházejí některé další distribuce, ať již nekomerční či komerční. Mezi tyto distribuce patří mj. Knoppix, Linspire, Ubuntu či Xandros.

Aktuální verze Debianu je 5.0 s kódovým označením Lenny. Počínaje verzí 6.0 bude mít Debian dvouletý vývojový cyklus - zmrazení nastane každý lichý rok v prosinci s tím, že nové vydání bude následovat v první polovině následujícího roku. Bezpečnostní aktualizace pro starou verzi vycházejí ještě rok po vydání nové verze, což dává správcům dostatek času pro upgrade na aktuální verzi. Verze 6.0 vyjde v první polovině příštího roku.

Z hlediska nasazení na server vyniká Debian svou stabilitou a relativně dlouhým vývojovým cyklem, který představuje jistý kompromis mezi distribucemi s dlouhou podporou jako CentOS nebo Ubuntu LTS, a distribucemi, které vycházejí každých 6-12 měsíců (Fedora, apod.). Jak už bylo zmíněno, Debian je jednou z nejrozsáhlejších distribucí - jeho repositáře čítají 25 tisíc balíčků, což nám dává ohromnou paletu pro výběr serverového softwaru k nasazení, o jehož aktualizaci se postará distribuce.

Delší vývojový cyklus může u Debianu naopak vadit tam, kde záleží na tom, aby byl software aktuální. Tohle do jisté míry řeší repositář Debian-backports, který obsahuje novější verze vybraného softwaru pro stabilní větev Debianu. V případě, že je aktuálnost softwaru důležitá, může být nicméně přesto vhodnější zvolit distribuci s kratším vývojovým cyklem (Fedora, apod.) nebo s kontinuálním vývojem (Gentoo, apod.).

## Správa linuxového serveru: RAID teoreticky

Ještě před tím, než se rozhodnete nainstalovat Debian na server, byste měli zvážit, jakým způsobem budete pracovat s pevnými disky. Mít všechno na jednom disku v jednom či více diskových oddílech je sice možné, ale existují i jiné možnosti, a sice RAID a LVM, potažmo ještě dm-crypt/LUKS pro diskové šifrování. V tomto dílu se budeme věnovat RAIDu.

### Kam se systémem a daty? RAID

O RAIDu zde již jeden pěkný článek vyšel - Když se řekne RAID... Nerad bych příliš opakoval to, co tu již zaznělo, a proto teorii omezím na nezbytné minimum a spíše se pokusím přidat několik důležitých praktických pohledů.

Co je RAID? Je to způsob práce se dvěma nebo více pevnými disky jako s jednou logickou datovou jednotkou (blokovým zařízením), která na fyzické úrovni poskytuje jistý kompromis mezi odolností proti výpadku jednoho nebo více disků, kapacitou a výkonem.

Odolnost proti výpadkům je zajišťována jistou formou redundance (liší se u jednotlivých typů RAIDu), která ovšem jistým způsobem snižuje úložný prostor (kapacitu) pole. Typů RAIDu je mnoho, já se zaměřím na ty nejtypičtější a nejzajímavější (informace o ostatních typech naleznete v článcích odkazovaných v závěru článku):

#### *RAID 0*

Data se rozmisťují střídavě po všech discích z pole, chybí redundance - ztráta jednoho disku znamená ztrátu všech dat z pole. Výkon je ze všech typů RAIDu nejlepší, kapacita pole je rovna počtu disků.

#### *RAID 1*

Zrcadlení. Data se zrcadlí na všechny disky z pole. Kapacita pole je rovna velikosti nejmenšího disku. Čtení z pole je rychlejší (lze číst najednou z více disků), zápis je pomalejší (zapisuje se na všechny disky současně). Pole udrží data při selhání n-1 disků (kde n je počet disků v poli).

#### *RAID 5*

Minimem jsou tři disky. Data a paritní informace se rozmisťují střídavě po všech discích z pole. RAID 5 má kapacitu n-1 disků a snese selhání jednoho disku. Čtení z pole je podobně rychlé jako u RAIDu 0, zápis je pomalejší a v tomto případě i náročný na výpočetní výkon (k datům se počítá parita).

#### *RAID 6*

Minimem jsou čtyři disky. RAID 6 je podobný RAIDu 5 s tím, že paritní blok není jeden, ale dva, takže pole, které má kapacitu n-2 disků, unese selhání kterýchkoliv dvou z nich. Výkon RAIDu 6 je podobný jako výkon RAIDu 5, náročnost na výpočetní výkon je ovšem o něco vyšší (počítají se dva paritní bloky).

#### *RAID 10*

RAID 10 je kombinací RAIDu 1 a 0 v tomto pořadí. Tj. nejprve se vytvoří dvě RAID 1 pole, a poté se nad nimi postaví RAID 0. Výhodou je dobrý poměr mezi odolností a výkonem.

#### *RAID 15 (nebo RAID 51)*

RAID 15 (nebo RAID 51) je kombinací RAIDu 1 a 5. Bývá označován jako RAID pro paranoidní, neb poskytuje poměrně velkou odolnost, za kterou se ovšem platí drastickým úbytkem kapacity.

### Praktické poznámky k RAIDu

#### *Jaký RAID vybrat?*

Záleží na tom, co máte k dispozici (počet disků/rozpočet) a co chcete získat (rychlost, odolnost proti výpadkům, apod.). Na menších serverech se velmi často používají typy 1, 5 a 10. Podstatným kritériem výběru je počet disků - u dvou použijete nejspíše typ 1, u třech a více budete vybírat mezi typy 5 a 10. Při větším počtu disků (nad 10) stojí za uváženou nahradit RAID 5 RAIDem 6, popřípadě zvolit pro vyšší odolnost RAID 51 nebo 61. Pro náš LAMP servřík plně postačí typ 1 (zrcadlení).

#### *Více disků = větší pravděpodobnost selhání*

Je jasné, že při propojení více pevných disků do diskového pole postupně vzrůstá pravděpodobnost selhání některého z disků. Z tohoto důvodu musíte při velkém počtu disků počítat s rizikem selhání více disků krátce po sobě, a upravit volbu typu RAIDu podle toho.

#### *Disky od různých výrobců*

Ideální je dávat do RAIDu různé disky (stejné kapacity) od různých výrobců (nejlépe ještě různě staré, ale to už bych asi chtěl trošku moc). Snížíte tak pravděpodobnost kritického selhání, pokud byste měli smůlu a nakoupili disky ze série, která má výrobní vadu (a kde by disky selhaly současně nebo krátce po sobě). Čas od času se každému výrobci disků "poštěstí" vydat sérii s nějakou více či méně kritickou vadou. Pokud nebudete kupovat disky od různých výrobců, je dobré koupit alespoň disky z různých sérií.

#### *Hardwarový, nebo softwarový RAID?*

Kvalitní hardwarový řadič s baterií zálohovanou vyrovnávací pamětí může být dobrým důvodem, proč nepoužít linuxový softwarový RAID. Levným RAID řadičům (zejména pak těm, které naleznete na desktopovém hardwaru) bych se raději vyhnul ve prospěch sofwarového RAIDu. V případě hardwarového RAIDu je dobré pamatovat na to, že RAID řadiče od různých výrobců spolu obvykle nejsou kompatibilní - vaše data tedy mohou být vázána na příslušný typ hardwarového řadiče od jistého výrobce (s tím, že pokud by se řadič rozbil, už byste se k datům bez náhradního řadiče nemuseli dostat).

#### *RAID není náhradou záloh!*

Redundance svádí k tomu považovat RAID za jistý typ zálohování. To je však velmi špatný přístup. Selhat může ledacos, počínaje více disky, než kolik vám kryje redundance, přes řadič či softwarový RAID až po operační systém, aplikaci, uživatele či administrátora. RAID je vhodným prostředkem pro zajištění odolnosti serveru proti výpadku pevného disku, ale zálohování nemůže nahradit.

#### *Co po výpadku disku?*

Při použití RAIDu je vhodné nejenom počítat s možným selháním disku, ale také zauvažovat o tom, co se stane pak. Mnohé typy RAIDů se používají tak, že sice snesou výpadek jednoho disku, ale pokud se tak stane, celé pole pak obvykle visí na vlásku - stačí selhání dalšího disku k tomu, abyste přišli o data.

Dovolte mi tento problém demonstrovat na příkladu. Řekněme, že jste si zvolili RAID 5 o třech discích a jeden vám selhal. V této situaci máte pole sice funkční, ale degradované, bez jakékoliv redundance. Pevné disky už za sebou nějaký čas mají, takže riziko dalšího selhání tu je, a ne úplně nepatrné. Výměna disku a opatření náhrady vám zabere nějaký čas. Ale ani po tom, co disk seženete a vyměníte, není vyhráno. Dokonce lze říci, že v té chvíli to nehorší teprve začne - bude třeba pole rekonstruovat, což obnáší přečtení všech nebo velké části dat ze zbývajících disků. Pokud při tomto procesu vypadne další disk, je po všem. Teprve ve chvíli, kdy bude nový disk zapojen a synchronizován se zbytkem pole, si můžete opravdu oddechnout.

Z tohoto hlediska nemusí být úplně od věci provést po selhání a těsně před vložením nového disku rozdílovou zálohu (pokud tedy používáte přírůstkové zálohování). O zálohování pojednává článekZálohujeme pomocí Dump/Restore.

#### *Spare disky*

Jednou z možností, jak zkrátit dobu mezi selháním disku, jeho výměnou a zařazením nového disku do pole, je použití náhradních (spare) disků. To jsou disky, které jsou sice zařazené v diskovém poli, ale nepoužívají se, alespoň do doby, než nějaký disk z pole selže. Poté se ihned spustí rekonstrukce pole s pomocí tohoto náhradního disku.

Dodávám, že vzhledem k předchozímu bodu to nemusí být to pravé ořechové úplně ve všech situacích.

#### *Integrita RAIDů 5 a 6*

Po výměně defektního disku za nový začíná rekonstrukce pole. To obnáší přečtení všech sektorů ze všech disků. Disky si ovšem někdy usmyslí, že některý sektor z nějakého důvodu nepřečtou (o tomhle pojednává lehce katastrofický článek Why RAID 5 stops working in 2009, který zmiňuji především kvůli zajímavým komentářům). Pokud se na takový sektor natrefí během rekonstrukce, vypadne obvykle příslušný disk z pole, a máme velký problém.

Důvodů, proč některý sektor nepůjde přečíst, může být celá řada, počínaje výpadkem proudu, když zrovna probíhá zápis, nebo prostě vznik špatného sektoru. Ano, moderní pevné disky mají oblast záložních (spare) sektorů, kam špatné sektory realokují. Jenomže, realokace špatného sektoru se provádí při zápisu, nikoliv při čtení (čtení se totiž může podařit některým z opakovaných pokusů). Tudíž nás realokace v tomto směru nezachrání.

Dalším možným problémem je situace, kdy se vlivem softwarové chyby či výpadku proudu při zápisu ztratí konzistence mezi daty a paritní informací. To sice nevyvolá chybu čtení či výpadek dalšího disku, dokonce i rekonstrukce úspěšně proběhne, ale zapíšou se špatná data (a co je horší - vy se nic nedozvíte).

Abyste těmto problémům do jisté míry předešli, je vhodné RAID 5 i RAID 6 podrobit čas od času kontrole. Některé hardwarové řadiče to provádí automaticky (doporučuji zkontrolovat), v případě softwarového RAIDu v Linuxu je to zatím záležitost, která vyžaduje trošku manuální práce - umístění následujícího příkazu do cronu a jeho periodické opakování v řádu dní až jednoho týdne:

```
echo "check" > /sys/block/raid_zarizeni/md/sync_action
```

***Data corruption (poškození dat)***

V rámci RAIDu se nijak neverifikují zapsaná data. To znamená, že RAID je zranitelný vůči poškození dat(třeba vlivem špatného kabelu nebo řadiče, chyby v RAM modulu, apod.).

S takovou situací už jsem se jednou setkal - u jednoho levného SATA řadiče docházelo k bitovým chybám při čtení/zápisu, následkem čehož v delším období došlo ke rozsáhlému poškození dat na (softwarovém) RAIDu 1, který jsem na něm provozoval. I proto zdůrazňuji, že RAID není náhradou zálohování.

Poškození dat je to nejhorší, s čím se lze setkat - je totiž obvykle nepostřehnutelné. V době, kdy se váš systém začne chovat divně nebo dojde k nějaké jiné události, která vás donutí dívat se tímto směrem, může být již pozdě (a co je horší - chyby v datech se mohou promítnout i do záloh). Tohle třeba pěkně řeší ZFS(souborový systém od Sunu), který, na rozdíl od tradičních linuxových souborových systémů jako ext3 či reiserfs, dohlíží na integritu dat. Bohužel, ZFS v Linuxu není a z licenčních důvodů asi ani nebude. Máme pouze BTRFS, který zatím, soudě podle projektové wiki, stále není určený pro produkční nasazení.

Z tohoto důvodu nelze než doporučit alternativní postupy - Tripwire, md5sum nebo podobné nástroje pro kontrolu integrity vybraných souborů.

***Monitorování pole***

Chcete-li být zodpovědní, RAID nemůžete nechat běžet bez nějaké formy monitorování, které vás informuje v případě nějaké významné události (zejména pak výpadku pevného disku). Měli byste si tedy vytvořit nějaký systém monitorování diskového pole, abyste byli včas varováni, že máte vyměnit disk nebo (v horším případě) obnovit data ze záloh.

V Linuxu existuje nástroj *mdadm*, který umožňuje nejenom provádět správu softwarového RAIDu, ale je schopen fungovat i jako démon, dohlížet nad softwarovým RAIDem a informovat vás (třeba e-mailem), že došlo k nějaké významné události. Jeho konfiguraci se budu věnovat po instalaci Debianu. V tuto chvíli vám postačí vědět, že diskové pole je třeba nějakým způsobem monitorovat.

## Správa linuxového serveru: LVM a diskové šifrování

V tomto díle si probereme LVM (Logical Volume Manager) a diskové šifrování v podobě dm-crypt/LUKS. Je to poslední z přípravných dílů před instalací Debianu.

### LVM

LVM (Logical Volume Manager) je systém pro správu logických oddílů, v Linuxu je implementován pomocídevice mapperu, mechanismu pro mapování jednoho blokového zařízení na jiné. Ptáte se, proč používat LVM místo klasických diskových oddílů? LVM umí řadu věcí, které s "klasickými" diskovými oddíly neuděláte. Kupříkladu - vyhradíte si diskový oddíl určité velikosti a posléze (za provozu) zjistíte, že velikost je nevyhovující - oddíl je buď zbytečně velký, a ubírá tak místo ostatním oddílům, nebo je příliš malý, a začne na něm brzy docházet místo. Logické "oddíly" (dále svazky) spravované pomocí LVM můžete vytvářet, rušit, měnit jejich velikost, pořizovat z nich snapshoty, ale také je přesouvat mezi fyzickými zařízeními, a to vše za běhu, bez nutnosti restartu.

Další vlastností LVM je relativní nezávislost logických svazků na fyzických zařízeních. Můžete "poskládat" LVM z několika oddílů na různých discích, a tuto "skupinu" pak rozdělit, jako by se jednalo o jeden velký spojitý prostor. Podobnost s RAIDem 0 je zde evidentní, stejně jako vyplývající hrozba ztráty dat v případě ztráty některého fyzického zařízení. LVM nicméně umí i ekvivalent RAIDu 1, tedy zrcadlení dat na více fyzických zařízení. Neumí už ovšem paritně založenou redundanci (ekvivalent RAIDu 5 a 6).

#### *Princip funkce LVM*

Představte si LVM jako vrstvu, kam na jedné straně nasypete "fyzická zařízení" (de facto jakékoliv myslitelné blokové zařízení), a na druhé straně si na vzniklém spojitém prostoru vytvoříte logické svazky, tedy z pohledu linuxového správce konkrétní bloková zařízení, na kterých pak můžete dle libosti vytvářet souborové systémy a se kterými můžete poměrně flexibilně manipulovat, a to i za běhu. LVM můžete samozřejmě postavit nad jakýmikoliv blokovými zařízeními, tedy třeba nad linuxovým softwarovým RAIDem nebo nad šifrovaným zařízením (třeba pomocí dm-crypt/LUKS, viz níže).


*Schéma ilustrující princip funkce LVM*

A teď ještě jednou a za pomoci terminologie LVM (vizte obrázek nad tímto odstavcem). Úplně vespod máme **physical volumes** (fyzické svazky). To mohou být diskové oddíly, celé pevné disky nebo úplně jiná bloková zařízení. Všechna tato zařízení se umístí do **volume group** (skupiny svazků), která se pak tváří jako jeden velký spojitý prostor, který můžete alokovat do konkrétních **logical volumes** (logických svazků). Důvodem, proč preferuji anglickou terminologii, je fakt, že příkazy pro správu LVM jsou z anglických názvů odvozeny. Kupříkladu, vytvoření fyzického svazku (**p**hysical **v**olume) se provádí příkazem pvcreate. Ale samotnou správu LVM si probereme podrobněji až v některém z dalších dílů.

#### *Smysl LVM na serveru*

Přínosem LVM je především flexibilita práce s logickými svazky ve vztahu k fyzickým zařízením. Pokud na nějakém svazku začne docházet místo, není problém ho zvětšit, a to i za běhu. Snapshoty jsou neocenitelnou pomůckou v mnoha situacích (počínaje snadnějším zálohováním). Naopak LVM představuje vrstvu funkcionality navíc, kde se může vyskytnout problém (i když je třeba dodat, že LVM v Linuxu je již dostatečně zralá technologie pro produkční nasazení) nebo která může zkomplikovat záchranu dat v případě havárie (opět je třeba zdůraznit nutnost zálohování).

### Diskové šifrování

Diskové šifrování na serverech je minimálně využívanou funkcionalitou, ale ať už se rozhodnete diskové šifrování nasadit či nenasadit, měli byste vědět, o co se jedná, co vám diskové šifrování nabídne, před čím vás ochrání, a před čím vás naopak neochrání.

Diskové šifrování je metoda, při níž se obsah pevného disku (resp. konkrétních šifrovaných oddílů) udržuje zašifrovaný, ale operační systém je schopen prostřednictvím online šifrování a dešifrování zpřístupnit šifrovaný svazek tak, jako by se jednalo o svazek nešifrovaný. Předpokladem je samozřejmě zadání správného hesla.

To znamená, že v optimálním případě nebude útočník schopen získat bez znalosti klíče citlivá data na šifrovaných oddílech, pokud si třeba odnese pevný disk nebo ho vytrhne z běžícího serveru.

Úskalí diskového šifrování je nicméně mnoho, a na serveru to platí dvojnásob. **Za prvé**, velmi záleží na použitých kryptografických metodách a jejich implementaci. Diskové šifrování je problematické z toho důvodu, že se šifrují data o velkém objemu, a ještě navíc data, která mají určité vzorce, a o kterých lze mnohé usoudit, i když data nejsme schopni dešifrovat (toto zahrnuje datové struktury souborových systémů, prázdné místo, apod.). Je tedy třeba použít vhodné řešení (v našem případě dm-crypt/LUKS), vhodné kryptografické metody (viz dále) a samozřejmě dostatečně silné heslo.

**Za druhé**, diskové šifrování v současné době není schopné nahradit fyzické zabezpečení serveru - to znamená, že před útočníkem schopným dostat se fyzicky k našemu serveru, data příliš chráněna nejsou. Existuje řada útoků, proti kterým se u serveru nelze rozumně bránit. Tyto útoky zahrnují nejobávanější cold boot attack (předpokládá fyzický přístup k serveru, když server běží), ale i útoky na části systému, které z principu šifrovat nelze (zavaděč, obrazy jader, iniciální ramdisk, apod.) - tyto útoky je možné realizovat i v situaci, kdy je server vypnutý.

**Za třetí**, diskové šifrování znamená poměrně citelný dopad na rychlost čtení a zápisu dat, ale také na zatížení serveru - šifrování a dešifrování totiž zajišťuje procesor, a šifrování / dešifrování jako takové je výpočetně náročné. V Linuxu je navíc ještě problém v tom, že diskové šifrování obstarává u každého oddílu jen jedno vlákno, což nám vytváří úzké hrdlo u vícejaderných a víceprocesorových konfigurací, kde není dost dobře možné jejich výhodu paralelních výpočtů využít.

**Za čtvrté**, stejně jako u LVM se jedná o vrstvu funkcionality navíc, kde může dojít k chybě, a kde může přehození jediného bitu způsobit nečitelnost celého jednoho bloku. Dvouúrovňové šifrování v rámci LUKS navíc vytváří *single point of failure*, kde, pokud dojde k narušení integrity hlavičky šifrovaného svazku, přijdeme zcela jistě o všechna data (hlavička totiž obsahuje klíč k dešifrování celého oddílu).

**Za páté**, pro dešifrování oddílů je nutné zadat heslo, což u serveru, který by měl po restartu co nejdříve naběhnout, komplikuje situaci. Ještě více situaci komplikuje absence fyzické přítomnosti správce, pokud je server umístěn v datacentru a spravován na dálku. Toto se však dá řešit zadáním hesla přes SSH. Není mi známo, že by Debian tuto možnost měl vestavěnou, ale je možné sestavit vlastní řešení třeba na základěnávodu na debian-administration.org.

Jaký má tedy diskové šifrování smysl? U serveru přichází v úvahu následující alternativy:

- ochrana proti úniku dat z defektních nebo jinak vyřazených pevných disků
- ochrana proti úniku dat následkem zcizení serveru - ovšem pouze pokud by ke zcizení došlo ve chvíli, kdy je server vypnutý, tzn. třeba při jeho převozu z jednoho datacentra do druhého
- minimální ochrana před neznalým útočníkem s fyzickým přístupem k disku

Uživatelé RAIDu mi nyní pravděpodobně řeknou, že k zabezpečení dat před prvním zmíněnou situací postačí fakt, že data se nachází třeba v RAIDu 5 nebo 6. Ano, v takové situaci sice nelze rekonstruovat původní data, nicméně vzhledem k tomu, že data se v rámci RAIDu 5 a 6 rozdělují na bloky určité velikosti, je přeci jen možné získat nějaká data i z disku v diskovém poli - tím, že se pokusíme vytáhnout informace z datových bloků. Pokud nakládáme s citlivými daty, může být i takto omezený "únik" nepřijatelný.

Diskové šifrování není vzhledem ke svým nevýhodám vhodné pro nasazení na běžný server, ale určitě se velmi hodí správcům pro ochranu citlivých dat jako SSH klíčů, hesel nebo jiných materiálů na přenosných zařízení nebo desktopech.

#### *dm-crypt*

Dm-crypt je vrstva využívající device mapper (viz výše), která umožňuje využít CryptoAPI (konkrétní implementace šifrovacích algoritmů, módů, hashí, apod.) v linuxovém jádře k online šifrování blokových zařízení.

#### *LUKS*

LUKS je postavený na dm-cryptu a jedná se o rozšíření, které poskytuje dvouúrovňové šifrování spolu s vhodnou funkcí pro derivaci klíče (PBKDF2). Dvouúrovňové šifrování znamená, že oddíl je šifrován hlavním klíčem (master key), a tento klíč je pak zašifrován některým z uživatelských klíčů. Uživatel zadá svůj klíč, kterým je dešifrován hlavní klíč, a ten dešifruje příslušný šifrovaný oddíl. Výhodou je možnost použít více klíčů nebo kompromitovaný klíč velmi snadno změnit, bez nutnosti celý oddíl přešifrovat. Funkce pro derivaci klíče PBKDF2 o něco málo zvyšuje bezpečnost slabších klíčů a znesnadňuje slovníkový útok.

***Praktická realizace diskového šifrování***

Jak už bylo zmíněno, nelze šifrovat úplně všechno, oddíl zahrnující /boot musí zůstat nešifrovaný. Všechno ostatní (včetně kořenového oddílu) může být šifrované (a má-li šifrování mít alespoň nějaký smysl, je třeba šifrovat nebo jinak zabezpečit swap i veškerá umístění, kam by se mohla dostat citlivá data - např. /tmp). Šifrování lze postavit nad RAIDem a výsledný šifrovaný svazek použít pro LVM, třeba jak je to naznačeno na tomto obrázku.

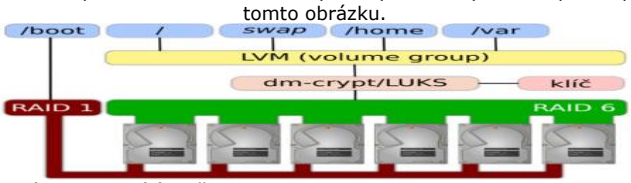



*Schéma ilustrující možnou konfiguraci RAIDu, dm-crypt/LUKS a LVM*

Na tomto obrázku bych rád ilustroval, jak složité struktury pro uložení a zabezpečení dat je možné vytvářet. Spolu s tím bych rád upozornil na to, že na pořadí v tomto případě velice záleží - proto je třeba navrhnout způsob práce s úložištěm s rozmyslem, což je také důvodem, proč se těmto tématům věnuji před samotnou instalací.

Co se týče nastavení, volby šifry a šifrovacího módu, jen ve stručnosti. Šifru si vyberte dle libosti, ale pokud možno takovou, která není zastaralá (jako například Blowfish) - ideální je vybírat některého z finalistů AES. U diskového šifrování dnes nepřipadá v úvahu jiný mód než XTS. Módy cbc-essiv i LRW obsahují zranitelnosti. Více o teorii šifrovacích módů pro diskové šifrování si můžete přečíst v článku Disk encryption theory.

To je pro tentokrát vše. Příště se konečně vrhneme na instalaci Debianu na server. Práci s LVM a dm-crypt/LUKS a jejich správu probereme v některém z dílů po instalaci.

## Správa linuxového serveru: Instalace

Jelikož čtenáři tohoto seriálu mají již jisté znalosti, nemyslím si, že je nutné procházet instalaci krok za krokem. Zaměřím se spíše na praktické záležitosti kolem instalačního procesu, upozorním na některé problémy a především osvětlím spektrum možností, které máte v případě instalace Debian GNU/Linuxu.

### Instalační média a možnosti instalace Debianu

Instalačních média Debianu lze rozdělit do dvou kategorií - klasická instalační média zahrnující 31 CD nebo 5 DVD obsahující veškerý software z hlavních repositářů a speciální instalační média zahrnující mj. minimalistické médium určené pro síťovou instalaci (netinst). O tom, jak je software umístěn do jednotlivých médií rozhoduje jeho popularita - nejpopulárnější software je na prvním médiu a nejméně populární na posledním. Máte-li k dispozici připojení na síť, není důvod nepoužít médium pro síťovou instalaci (iso obraz má pod 200MB). Postačí samozřejmě i první "klasické" instalační CD. Více není důvod stahovat. Debian lze nainstalovat i jinými způsoby, třeba z disket, flash disku nebo pomocí bootování ze sítě. Nainstalovat Debian je možné i z jiné distribuce (nebo ručně z instalačního média - více viz níže).

Instalátor Debianu má klasickou textovou verzi, ale také grafickou, která je funkčně prakticky totožná. Pokud budete instalovat Debian 5.0, a budete chtít pomocí instalátoru vytvořit LVM, RAID, dm-crypt/LUKS, zvolte raději grafický instalátor. Textový instalátor má tendenci v tomto vydání padat o dost více než ten grafický.

### Rozdělení disku

U desktopu si vystačíte i s jediným oddílem vyhrazeným pro Linux, u serveru bývá dobré vytvořit více oddílů a některé adresáře od sebe oddělit. Pokud budete využívat LVM, RAID nebo dm-crypt/LUKS, pak budete muset vytvořit kromě kořenového oddílu samostatný oddíl pro /boot. Je možné mít /boot oddíl zrcadlený pomocí linuxového softwarového RAIDu 1 na více disků. Pokud budete vytvářet dané pole ručně, dejte si pozor, ať ho vytvoříte tak, aby bylo možné příslušný souborový systém /boot přimountovat přímo z diskového oddílu (tedy ne z pole, ale z jednoho disku v poli). Bez toho nebude GRUB schopen se ani nainstalovat, natož bootovat (bude hlásit chybu č. 17). Kromě /boot je vhodné osamostatnit data od zbytku systému, pokud možno zahrnující jak webové aplikace, tak domovské adresáře uživatelů, databázi, poštu, apod. - toto vše je možné buď přesunout do jediného adresáře (obvykle /home), nebo rozdělit do více oddílů (třeba i s různými souborovými systémy). Velmi vhodné je též osamostatnit adresář /var/log, který mívá tendenci v určitých situacích nadměrně růst. Pokud bude na samostatném oddílu, budete mít o starost méně, protože pokud by náhodou "přetekl", neohrozí vám to zbytek systému (nedostatek místa v některých adresářích může způsobit velké problémy).
Je možné osamostatnit i adresář /var, kde se nachází různá "proměnlivá" data, mj. třeba cache správce balíčků (/var/cache/apt), data z databází (MySQL, PostgreSQL, apod.), ale také kritické systémové adresáře jako /var/run, kam se zaznamenávají informace o běžících démonech, a kde by v žádném případě nemělo dojít místo.
Samostatný oddíl pro swap je též velmi vhodný. Samotná velikost swap oddílu by se měla pohybovat v rozumných mezích vzhledem k dostupné paměti na daném serveru (dle některých zdrojů postačí polovina RAM, jiné zdroje uvádějí sofistikovanější pravidla pro určení velikosti swapu). Obecně, vytvářet příliš velký swap je zbytečné, ale stejně tak není dobré na swap úplně zapomenout. Je třeba si uvědomit, že server běží obvykle velmi dlouho v kuse, software na něm běžící si ukládá data do paměti, ale ne vždy k nim potřebuje často přistupovat. Oblasti operační paměti, které nejsou delší dobu využívány, tak mohou být odloženy do swapu, čímž zůstane více operační paměti pro ty aplikace, které ji potřebují, nebo pro diskovou cache.
Adresář /tmp může využívat buď *tmpfs*, což je virtuální souborový systém, který místo pevného disku využívá operační paměť (nebo swap), nebo pro něj vyčlenit samostatný oddíl. Do /tmp má právo zapisovat kdokoliv, tudíž je vhodné počítat s možným zaplněním tohoto oddílu a následky z toho plynoucími (bohužel, samotné zaplnění /tmp bude s některými aplikacemi či démony dělat psí kusy). Pokud budete využívat*tmpfs*, jistě mu nezapomeňte specifikovat maximální velikost.

### Paměť na serveru

Na serveru je nutné sledovat využití paměti a ve chvíli, kdy začne paměť docházet, je třeba dokoupit novou (nebo situaci řešit jinak). Pokud systém vyčerpá celou paměť, nebo téměř celou paměť, začne docházet ke dvěma věcem. Systém se kriticky zpomalí tím, že začne přesouvat data z paměti do swapu a naopak (trashing) a pokud paměť skutečně dojde (včetně swapu, nebo pokud swap není), pak se nastartujeoom_killer. To je procedura jádra, která uvolní kus paměti tím, že sestřelí nějaký běžící proces. Rozhoduje se podle určitých kritérií, ovšem ne vždy se jí podaří situaci správně ohodnotit. Není možné spoléhat na to, že oom_killer úspěšně sestřelí nějaký splašený proces, aniž by ohrozil procesy jiné. Pro lepší představu, v dokumentaci k příslušné proceduře se píše:

```
/*
* oom_badness - calculate a numeric value for how bad this task has been
* @p: task struct of which task we should calculate
* @p: current uptime in seconds
*
* The formula used is relatively simple and documented inline in the
* function. The main rationale is that we want to select a good task
* to kill when we run out of memory.
*
* Good in this context means that:
* 1) we lose the minimum amount of work done
* 2) we recover a large amount of memory
* 3) we don't kill anything innocent of eating tons of memory
* 4) we want to kill the minimum amount of processes (one)
* 5) we try to kill the process the user expects us to kill, this
*    algorithm has been meticulously tuned to meet the principle
*    of least surprise ... (be careful when you change it)
*/
```

### Rootovské heslo

Jelikož nejsme na desktopu, ale na serveru, kde je třeba být přeci jen zodpovědnější než na desktopu, dodávám, že heslo pro roota byste měli zvolit pečlivě a s ohledem na obecné zásady pro tvorbubezpečných hesel.
Dodám jen jeden drobný postřeh z praxe - v heslech je dobré se obloukem vyhnout diakritice. Speciální znaky jsou v pořádku (hvězdička, tečka, pomlčka, apod.), ale s diakritikou obvykle bývá celá řada problémů a člověka ne vždy napadne, že problém by mohl být zrovna tady.

### Přizpůsobení instalátoru a manuální instalace

Proces instalace v rámci instalátoru Debianu je možné různými způsoby ovlivňovat. Především je možné se kdykoliv vrátit k některému z předchozích kroků a cokoliv změnit pomocí klávesy [Esc]. Druhou zajímavou možností je "expertní režim", který je možný navolit v boot menu instalačního média přes Advanced options | Expert install. Tento režim umožňuje ovlivnit řadu parametrů, které si instalátor jinak řídí sám. Ani to ale někdy nemusí pomoci, a proto je dobré vědět, že je možné provést i plnou ruční instalaci v příkazové řádce. V takovém případě budete postupovat velmi podobně jako s instalátorem, jen ručně (nebo i kus s pomocí instalátoru) - zprovozníte síť (a připojení k Internetu), připravíte si oddíly, namountujete je někam do adresářové struktury a využijete nástroje *debootstrap* k instalaci základu systému do připraveného adresáře, takto:

```
debootstrap lenny /mnt/new_system
```

Poté je třeba upravit některé konfigurační soubory jako /etc/fstab, nakonfigurovat zavaděč (LILO nebo GRUB) a nainstalovat jej na správné místo (obvykle MBR).

### Minimální instalace

U serveru je vhodné nainstalovat co nejmenší počet balíčků nutný k tomu, abyste zajistili jeho optimální funkci. Už jen z bezpečnostního hlediska - čím méně běžících procesů, čím méně balíčků, čím méně kódu, který bude na serveru aktivně využíván, tím menší pravděpodobnost výskytu nějaké zranitelnosti.
Debian umí provést minimální instalaci, kdy nainstaluje skutečně pouze základ systému s tím, že si pak vy jako správci doinstalujete to, co potřebujete (a jen to, co potřebujete). Při instalaci stačí ve fázi, kdy se vás instalátor ptá na software, který má nainstalovat, odškrtnout obě výchozí zvolené volby (**Desktop system** a**Standard installation**). Tento postup doporučuji, pokud budete instalovat nový server.
A tímto bych tento díl ukončil s tím, že příště proberu základní poinstalační aktivity a pozvu vás na menší exkurzi do vašeho nového systému.

## Správa linuxového serveru: Co dělat po instalaci

V tomto díle proberu základní poinstalační aktivity a pozvu vás na menší exkurzi do nového systému Debian GNU/Linux. Dozvíte se o několika užitečných nástrojích pro administrátora linuxového serveru.

### Instalace některých důležitých balíčků

Těsně po instalaci je vhodné nainstalovat některé balíčky, které vám jako správcům usnadní práci. Já obvykle instaluji tyto balíčky:

- **openssh-server** - SSH server pro vzdálenou správu
- **mc** - terminálový panelový správce souborů (Midnight commander)
- **vim-nox** - editor Vim (bez závislosti na X)
- **screen** - správce sezení v terminálu
- **htop** - interaktivní správce úloh
- **lshw** - jeden z nejlepších nástrojů pro získávání informací o hardwaru
- **iotop** - sledování I/O operací procesů v reálném čase
- **apg** - řádkový generátor hesel

Po instalaci se určitě hodí nainstalovat SSH server, abyste mohli k serveru přistupovat vzdáleně a nemuseli být fyzicky u něj. Zabezpečení SSH serveru se budeme věnovat podrobně v některém z dalších dílů, prozatím postačí držet se několika rad:

- v `/etc/ssh/sshd_config` nastavit volbu `PermitRootLogin` na jinou hodnotu než `yes` (nebo zvolit opravdu velmi silné heslo pro roota) - v úvahu přichází hodnota `no` (zakáže vzdálené přihlášení na roota) nebo hodnota `without-password` (povolí vzdálené přihlášení roota pouze SSH klíčem)
- ujistit se, že v systému není žádný testovací uživatel se slabým heslem (uživatel `test` s heslem `test`zajistí téměř spolehlivě kompromitaci serveru útočníkem v řádu hodin)
- používat ke vzdálenému přístupu uživatelský účet, který jste si vytvořili při instalaci, a opatřit jejbezpečným heslem
- ideální je hlásit se pouze pomocí SSH klíčů a nastavit v `/etc/ssh/sshd_config` volbu`PasswordAuthentication` na `no`, čímž zakážete přihlašování heslem pro všechny uživatele

Midnight Commander je výborným pomocníkem nejenom pro začínající správce. Spolu s ním získáte i uživatelsky přívětivý textový editor mcedit. Jako editor můžete samozřejmě používat i něco jiného, třeba můj oblíbený Vim (Vi Improved). I když se vám do toho nemusí chtít, naučit se základy práce s editorem Vi není pro správce unixových serverů vůbec na škodu, protože s některou z jeho variant se zajisté setkáte u drtivé většiny serverů, které třeba dostanete pod svou správu. Naopak jiné editory nemusíte mít mnohde k dispozici.

Pokud potřebujete provést nějakou časově náročnější operaci, provádět více operací současně, nebo pokud se připojujete přes ne zcela spolehlivé připojení, hodí se vám nástroj Screen, který vám zajistí jednak možnost spouštět více "virtuálních terminálů" v jediném SSH sezení, ale který také zajistí kontinuitu vaší práce, pokud se třeba SSH spojení přeruší. Screen vám v takovém případě zůstane běžet na pozadí a vy se k němu můžete kdykoliv znovu "připojit" příkazem:

```
screen -x
```

Htop je trošku příjemnější nástroj pro správu procesů než klasický top, používá barvy a je trošku intuitivnější. Občas se vám pro diagnostiku hodí i nástroj iotop, který vám řekne, jaký proces zrovna provádí práci s disky a jak moc je zatěžuje.

Nástroj lshw je nedocenitelný pomocník, pokud se chcete dozvědět více o hardwaru příslušného serveru. Umí toho zjistit neuvěřitelně mnoho, třeba i osazení jednotlivých paměťových bank. Vše umí prezentovat v jasné hierarchické struktuře a dokonce zvládá i export informací do HTML nebo XML.

Pro rychlou změnu nebo nastavení hesel jsem zvyklý instalovat nástroj apg, pomocí kterého je možné generovat hesla přímo na serveru.

### Exkurze do nového systému

Jako správci byste měli mít přehled o nejdůležitějších souborech, adresářích a skriptech v Debianu. Předesílám, že řada z těchto umístění je totožná i v případě jiných distribucí:

#### *Stěžejní konfigurační soubory*

- `/etc/fstab` - nastavení připojení souborových systémů do jednotlivých adresářů
- `/etc/crypttab` - nastavení šifrovaných souborových systémů
- `/etc/network/interfaces` - nastavení sítě
- `/etc/resolv.conf` - nastavení DNS serverů pro systémový resolver
- `/etc/sysctl.conf` - "permanentní" nastavení parametrů jádra
- `/etc/security/limits.conf` - nastavení limitů pro uživatele
- `/etc/ssh/sshd_config` - konfigurace SSH démona (viz výše)
- `/etc/apt/sources.list` - konfigurace repositářů správce balíčků Debianu
- `/etc/pam.d/*` - konfigurace PAMu (zásuvné autentikační moduly)

Většina těchto konfiguračních souborů má buď vestavěnou nápovědu, nebo manuálovou stránku. V dalších dílech si některé z nich představíme blíže.

#### *Stěžejní skripty a umístění*

Jak víte, po spuštění počítače a provedení příslušných testů předá BIOS řízení zavaděči (maličký program v prvních 446 bytech MBR). Zavaděč nahraje do paměti obraz jádra (popř. ještě iniciálního ramdisku) a spustí jej. Jádro připojí kořenový souborový systém a spustí /sbin/init, který začne pouštět další služby (démony) a procedury, jejichž obslužné skripty se nacházejí v /etc/init.d.

V Linuxu je možné nadefinovat různé úrovně běhu (runlevel), v rámci kterých se spouští (nebo vypínají) různé služby. Úrovně běhu v Debianu jsou ve výchozím stavu nadefinované pouze dvě, a to úroveň 1, tedy jednouživatelský režim bez sítě, a úroveň 2, kde běží naopak úplně všechno. Pro úplnost dodávám, že existují ještě dvě další, výchozí úrovně běhu, úroveň 0, vypnutí systému (shutdown), a úroveň 6, restart.

O tom, jaké služby se ve které úrovni běhu spustí nebo vypnou, rozhodují symbolické odkazy v adresářích/etc/rc?.d. Na změny v těchto adresářích lze použít nástroj update-rc.d. Výchozí úroveň běhu a další typy služeb a procedur, které spouští init, se konfigurují v souboru /etc/inittab.

Jako správcům se vám bude určitě hodit adresář /usr/share/doc, který obsahuje dokumentaci k jednotlivým nainstalovaným balíčkům. Zde najdete různé README, poznámky správců daných balíků, ale třeba i příklady konfiguračních souborů, apod.

Tím bych exkurzi i tento díl seriálu ukončil. Příště se vrátím k tématu RAID z praktického pohledu a ukážu vám, jak softwarový RAID v Linuxu funguje, jak jej vytvořit a spravovat

## Správa linuxového serveru: Softwarový RAID prakticky

Co je to RAID, jaké jsou jeho typy a pro který byste se měli rozhodnout, jste se již dozvěděli v minulých dílech. Nyní se na něj podíváme z praktického hlediska. Vytváření nového a modifikace stávajícího pole pro vás bude hračkou.

O RAIDu teoreticky: Správa linuxového serveru: RAID teoreticky

### Průzkum existujícího pole

Informace o všech diskových polích spravovaných Linuxem naleznete v souboru /proc/mdstat. Výpis tohoto souboru může vypadat třeba takto:

```
debian:~# cat /proc/mdstat
Personalities : [raid1]
md2 : active raid1 sdb2[0] sda2[1]
975763866 blocks super 1.2 [2/2] [UU]
bitmap: 5/466 pages [20KB], 1024KB chunk

md0 : active raid1 sdb1[0] sda1[1]
995904 blocks [2/2] [UU]

unused devices: &ls;none>
```

V tomto výpisu jsou vidět dvě pole, RAID 1 pole md0 o přibližné velikosti 1GB složené ze zařízení sdb1 a sda1a RAID 1 pole md2 o přibližné velikosti 1TB složené ze zařízení sdb2 a sda2. Obě dvě pole jsou aktivní se dvěma ze dvou zařízení (viz [2/2]).

Zkusme se podívat na jiný výpis:

```
debian:~# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 hda1[0] hdd1[3](F) hdb1[1]
1172608 blocks [3/2] [UU_]

unused devices: &ls;none>
```

Zde je patrný problém se zařízením hdd1, které bylo označeno jako poruchové. Proto je u něj písmeno **F** (f jako faulty). Pole je aktivní, ale pouze se dvěma ze tří zařízení (viz [3/2]). Povšimněte si ve výpisu posloupnosti označující stav jednotlivých zařízení seřazených podle jejich pořadí v poli - [UU_]. Zde značí písmeno U aktivní zařízení v poli a podtržítko značí chybějící aktivní zařízení.

V momentě, kdy chybné zařízení vyřadíme z pole a zařadíme jeho náhradu, začne rekonstrukce pole:

```
debian:~# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 hdd1[3] hda1[0] hdb1[1]
1172608 blocks [3/2] [UU_]
[=====>.............]  recovery = 28.0% (328768/1172608) finish=0.7min speed=18264K/sec

unused devices: &ls;none>
```

Detailnější výpis informací o diskovém poli můžeme získat pomocí nástroje mdadm:

```
debian:~# mdadm -D /dev/md2
/dev/md2:
Version : 01.02
Creation Time : Mon Nov  9 13:53:36 2009
Raid Level : raid1
Array Size : 975763866 (930.56 GiB 999.18 GB)
Used Dev Size : 1951527732 (1861.12 GiB 1998.36 GB)
Raid Devices : 2
Total Devices : 2
Preferred Minor : 2
Persistence : Superblock is persistent

Intent Bitmap : Internal

Update Time : Sat Jan  9 16:07:04 2010
State : active
Active Devices : 2
Working Devices : 2
Failed Devices : 0
Spare Devices : 0

Name : 'debian':2
UUID : 2e2fd27c:aa6c8143:b3829f5b:2e325dd8
Events : 12

Number   Major   Minor   RaidDevice State
0       8       18        0      active sync   /dev/sdb2
1       8        2        1      active sync   /dev/sda2
```

Pomocí nástroje mdadm můžeme zkoumat i jednotlivá zařízení v poli:

```
debian:~# mdadm -E /dev/sda2
/dev/sda2:
Magic : a92b4efc
Version : 1.2
Feature Map : 0x1
Array UUID : 2e2fd27c:aa6c8143:b3829f5b:2e325dd8
Name : 'debian':2
Creation Time : Mon Nov  9 13:53:36 2009
Raid Level : raid1
Raid Devices : 2

Avail Dev Size : 1951527733 (930.56 GiB 999.18 GB)
Array Size : 1951527732 (930.56 GiB 999.18 GB)
Used Dev Size : 1951527732 (930.56 GiB 999.18 GB)
Data Offset : 272 sectors
Super Offset : 8 sectors
State : clean
Device UUID : 998eca59:96570811:c1661b8e:93707a76

Internal Bitmap : 8 sectors from superblock
Update Time : Sat Jan  9 16:09:05 2010
```

Checksum : 3860e005 - correct
Events : 12

Array Slot : 1 (0, 1)
Array State : uU

**Vytvoření nového pole**

Veškerá správa linuxového softwarového RAIDu se provádí pomocí nástroje mdadm. Pokud byste chtěli vytvořit RAID 5 pole se třemi zařízeními (sda1, sdb1 a sdc1) a jednou náhradou (spare) v podobě zařízenísdd1, můžete použít následující příkaz:

```
mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sda1 /dev/sdb1 /dev/sdc1 --spare-devices=1 /dev/sdd1
```

Může se stát, že při sestavování pole nebudete mít hned k dispozici všechny disky, v takovém případě můžete místo chybějícího disku použít klíčové slovo missing (chybějící):

```
mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sda1 missing /dev/sdc1 --spare-devices=1 /dev/sdd1
```

Ačkoliv je možné sestavit pole v degradovaném režimu (s chybějícími redundantními aktivními zařízeními), tento postup se nedoporučuje. Zejména pak, pokud potřebujete na takové pole přesunout data ještě před tím, než doplníte zbývající zařízení.

**Modifikace existujícího pole**

Zařízení lze z pole odebírat a nová zařízení do pole vkládat. Vložení nového zařízení do pole je velmi jednoduché:

```
mdadm --manage /dev/md0 --add /dev/sde1
```

Každé pole má určitý definovaný počet aktivních zařízení (raid devices). Tento počet se nastavuje při vytvoření pole. Pokud je tento počet vyšší než momentální počet aktivních zařízení v poli (tj. nějaké aktivní zařízení v poli chybí, a pole je tudíž degradované), použije se nově přidané zařízení jako aktivní zařízení v poli, tedy jako náhrada za chybějící aktivní zařízení. V takové situaci dojde po zařazení daného zařízení do pole k zahájení jeho rekonstrukce.

Pokud je aktuální počet aktivních zařízení roven počtu aktivních zařízení v poli (žádné aktivní zařízení nechybí, pole tedy není degradované), pak se použije jako náhrada (spare). Náhradní zařízení sice v poli figurují, ale nejsou aktivní (nejsou na nich data). Teprve pokud některé z aktivních zařízení selže, pak bude náhrada automaticky zařazena do pole jako aktivní zařízení a začne jeho rekonstrukce.

Průběh rekonstrukce můžete sledovat třeba příkazem:

```
watch -n 1 'cat /proc/mdstat'
```

Odebrání zařízení z pole je trošku složitější. Záleží na tom, jestli je příslušné zařízení aktivní nebo ne. Pokud není aktivní (bylo označeno jako vadné nebo se jedná o náhradní zařízení), pak jej lze odebrat rovnou. Pokud se jedná o aktivní zařízení, je třeba jej nejdříve vyřadit z pole, což lze provést tak, že jej označíte jako vadné:

```
mdadm --manage /dev/md0 --set-faulty /dev/sde1
```

A následně je možné jej odebrat:

```
mdadm --manage /dev/md0 --remove /dev/sde1
```

Tím bych tento díl ukončil. Příště se podíváme na pokročilejší témata správy diskových polí jako řešení krizových situací, monitorování pole, atd.

## Správa linuxového serveru: Softwarový RAID prakticky (pokračování)

V tomto díle se podíváme na růst pole, řešení krizových situací, write-intent bitmap, scrubbing diskového pole, monitorování a bootování z pole.

### Růst pole

Růst pole (grow) je operace, při které se zvýší definovaný počet aktivních zařízení v poli. Tato operace je jednorázová a nevratná. Počet aktivních zařízení v poli již nelze snížit.

Růst pole má dvě fáze. Nejprve je třeba do pole přidat nové zařízení, o které má pole narůst:

```
mdadm --manage /dev/md0 --add /dev/sde1
```

Následně se provede růst:

```
mdadm --grow /dev/md0 --raid-devices=4
```

Růst lze provést pouze na čistém poli, tzn. pole nesmí být degradované.

### Řešení krizových situací

V situaci, kdy dojde k vyřazení jednoho nebo více disků z pole, je třeba zachovat rozvahu a nejprve popřemýšlet. Pokud přijdete o redundanci (tzn. selhání dalšího zařízení z pole znamená ztrátu dat), pak je vhodnější před zahájením rekonstrukce pole provést zálohu dat. Samotná rekonstrukce zbývající disky podstatně zatíží a jediná neopravitelná chyba čtení na některém z disků může situaci velice zkomplikovat.

Samotná výměna vadného disku je poměrně jednoduchá, postup byl naznačen v minulém díle. Zařízení je obvykle již jádrem označeno jako vadné (faulty), takže postačí jej odebrat z pole a nové, funkční zařízení do pole zařadit:

```
mdadm --manage /dev/md0 --remove /dev/sda1
mdadm --manage /dev/md0 --add /dev/sdf1
```

Pokud víte, že disk je vadný, nebo očekáváte jeho brzké selhání, ale jádro ještě disk považuje za funkční, musíte jej za defektní nejprve označit, než se pustíme do jeho odebrání z pole:

```
mdadm --manage /dev/md0 --set-faulty /dev/sda1
```

Znovu raději zopakuji, že pro případ katastrofálního selhání, kdy selže krátce po sobě více disků než je pole schopné unést, nebo i pro případ chyby administrátora, atd., je nutné nebrat disková pole jako náhradu zálohování. Důležitá data je nutné pravidelně zálohovat nejlépe na nějaké geograficky odloučené místo.

V případě, že se dostanete do nějaké opravdu svízelné situace, kdy třeba pole nepůjde znovu sestavit, doporučuji se kromě rozvahy podívat do archívu e-mailové konference linuxového softwarového RAIDu. Tam také naleznete dostatek motivace k tomu, abyste začali svá důležitá data zálohovat, pokud to ještě neděláte.

### Write-intent bitmap

Někdy se může stát, že některé aktivní zařízení v diskovém poli vypadne ze synchronizace s ostatními. To nemusí nutně znamenat, že disk je vadný, může se to stát třeba při nekorektním ukončení systému (způsobeném třeba výpadkem proudu), při problémech s kabeláží nebo s řadičem. Pokud k tomu dojde, je nutné pole opět zrekonstruovat, což obvykle trvá velmi dlouho.

Write-intent bitmap lze chápat podobně jako žurnál u souborových systémů - jádro se může takovém případě vyhnout plné synchronizaci, a synchronizovat pouze změněné oblasti, čímž se rekonstrukce podstatně zkrátí. Předpokladem je samozřejmě to, že do pole vracíme disk, který v něm již byl.

Pro write-intent bitmap je možné použít soubor (dle dokumentace však pouze na souborových systémech ext2 a ext3) nebo parametr "internal", který bitmapu uloží přímo na zařízení.

Skutečnost, že chcete použít write-intent bitmap je možné specifikovat již při vytváření pole parametrem -b. Pokud chcete write-intent bitmap vytvořit u již sestaveného pole, musíte použít režim "grow":

```
mdadm --grow /dev/md0 -b internal
```

Tímto příkazem byla vytvořena interní write-intent bitmap. Pokud se pak podíváte na /proc/mdstat, měli byste u příslušného zařízení vidět položku "bitmap":

```
debian:~# cat /proc/mdstat
Personalities : [raid1] [raid6] [raid5] [raid4]
md1 : active (auto-read-only) raid5 hda2[0] hdc2[3] hdd2[2] hdb2[1]
2939520 blocks level 5, 64k chunk, algorithm 2 [4/4] [UUUU]

md0 : active raid1 hdb1[1] hdc1[2] hda1[0]
1172608 blocks [3/3] [UUU]
bitmap: 144/144 pages [576KB], 4KB chunk
```

Pokud se z nějakého důvodu rozhodnete, že použijete nějaký soubor místo bitmapy interní, pak si dejte veliký pozor, aby to byl soubor na souborovém systému, který leží mimo dané pole.

### Scrubbing diskového pole

Jelikož různé oblasti pevných disků jsou čteny a zapisovány s různou četností, může dojít k situaci, kdy některý sektor na disku v poli přestane být čitelný, aniž bychom si toho byli vědomi. Následný pokus o přečtení takového sektoru, který může proběhnout i za dlouho, vyvolá neopravitelnou chybu čtení (uncorrectable read error), která vyřadí daný disk z pole.

Nyní uvažte situaci, kdy vám selže nějaký disk, a na některém ze zbývajících disků zůstala tato nášlapná mina v podobě nečitelného sektoru. Po vložení nového disku do pole a zahájení jeho rekonstrukce dojde na čtení z tohoto sektoru, což vyřadí další disk z pole. Pokud máte pouze jeden redundantní disk, pak jste se právě dostali do velmi nepříjemné situace.

Tomuto problému předcházejí hardwarové řadiče tím, že čas od času pole zkontrolují (všechno včetně redundance přečtou). Linuxový softwarový RAID toto nedělá automaticky, ale je možné mu to pomocí cronu nařídit. Bývá dobré to provádět spíše častěji, tedy nejméně jednou týdně.

Samotný příkaz, který scrubbing spustí, vypadá takto (předpokládám pole md0):

```
echo "check" > /sys/block/md0/md/sync_action
```

Pokud vytvoříte bash skript s tímto příkazem (nebo odpovídajícími příkazy pro zbývající disková pole), nastavíte mu právo pro spuštění a umístíte do /etc/cron.weekly, zajistíte jeho automatické týdenní spouštění.

### Monitorování pole

Disková pole je možné monitorovat stejným nástrojem (mdadm) jako pro jejich správu. Tento nástroj umí pracovat jako démon a hlásit události týkající se diskových polí. Jeho konfigurační soubor je/etc/mdadm/mdadm.conf, samotný démon by měl už rovnou běžet, pokud jste nainstalovali balíček mdadm.

Manuálová stránka konfigurační soubor velmi dobře popisuje, já tu zmíním dvě zajímavé volby:

```
MAILADDR mail@example.com,root@localhost
PROGRAM /root/skript_spusteny_pri_udalosti
```

Kromě obligátní e-mailové adresy, na kterou budou chodit e-maily s událostmi (předpokladem je samozřejmě funkční poštovní server), je možné specifikovat program, který mdadm při nějaké události spustí. Tomuto programu pak budou předány parametry zahrnující událost, zařízení odpovídající diskovému poli a, pokud to bude relevantní, označení disku v poli, kterého se událost týká.

### Bootování z pole

Do doby, než budeme mít běžně k dispozici GRUB 2, lze bootovat pouze z RAIDu 1, a to ještě ne ve všech případech. Zjistil jsem, že některá speciální nastavení (jiná verze superbloku, apod.) mohou bootování z RAIDu 1 zamezit. Pokud budete sestavovat pole se specifickými parametry, vyzkoušejte si, je-li možné namountovat souborový systém z jehdnoho z disků (či oddílů) v daném poli. Pokud se to nepovede, pak z takového pole nepůjde nabootovat.

Chcete-li primárně vytvořit jiné pole než RAID 1 pro data, pak doporučuji vytvořit na všech discích malý oddíl (kolem 1GB maximálně), na kterém bude v RAIDu 1 oddíl s adresářem /boot. GRUB i LILO zvládnout nabootovat z RAIDu 1, a zbytek místa na discích můžete použít k vytvoření jiného typu pole.

Tímto bych tento díl ukončil. Příště nakousnu problematiku zdraví pevných disků a v souvislosti s tím technologii S.M.A.R.T.

## Správa linuxového serveru: S.M.A.R.T. a zdraví pevných disků

Pevné disky obsahují obvykle to nejcennější - vaše data. Ať už využíváte linuxového softwarového RAIDu nebo ne, hlídat zdraví disků je určitě velmi důležité, stejně jako mít možnost odhadnout, v jakém stavu je disk, který se vám zrovna dostal pod ruku. A tomu se budu věnovat v tomto a v následujících dílech.

Pevné disky jsou v dnešní době relativně uzavřené systémy s vlastní vnitřní logikou, ale také vnitřním monitorováním a diagnostikou, a to je právě S.M.A.R.T. (*Self Monitoring, Analysis and Reporting Technology*). Původní myšlenkou této technologie byla schopnost předpovědět selhání disku a varovat uživatele ještě před tím, než k tomu dojde.

Realita je taková, že S.M.A.R.T. dokáže předpovědět blížící se selhání pevného disku jen v některých případech. Přesné procento případů udávají různé zdroje různě (Wikipedie udává 64 %, ale zdroj, ze kterého byla tato informace převzata, udává 30 %). Je jasné, že už jen z podstaty věci může S.M.A.R.T. předpovědět jenom takové typy poruch, které se projevují časem, pozvolna, nikoliv poruchy, které nastanou okamžitě a bez varování. To ale neznamená, že nemá cenu tuto technologii využít, naopak - i když není dokonalá, dokáže nám v řadě případů pomoci.

### Fungování S.M.A.R.T.

S.M.A.R.T. dělá tři věci. Jednak shromažďuje řadu údajů, které mají větší či menší vypovídací hodnotu o možném budoucím selhání. Pokud zjistí nějakou chybu při práci s diskem, pak ji zaznamená do svého vnitřního logu. Poslední věc, kterou S.M.A.R.T. umí, je testování disků na povel, tedy jakási vnitřní samodiagnostika.

### smartmontools

Balíček, který umožňuje využívat technologii S.M.A.R.T. v GNU/Linuxu se nazývá smartmontools. Naleznete ho v repositářích Debianu, ale i mnoha jiných distribucí.

To nejjednodušší, co můžete udělat, je přečíst údaje z konkrétního pevného disku, což provedete příkazem:

```
smartctl -a /dev/sda | less
```

Výpis vás asi na první pohled odradí, protože bude velmi dlouhý (proto také doporučuji použít less nebomore). Výpis nemusíte číst celý, stačí se zaměřit na ty nejpodstatnější části. První takovou částí je výpis hodnot atributů, které S.M.A.R.T. měří:

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000f   200   200   051    Pre-fail  Always       -       0
  3 Spin_Up_Time            0x0003   187   186   021    Pre-fail  Always       -       1616
  4 Start_Stop_Count        0x0032   100   100   000    Old_age   Always       -       577
  5 Reallocated_Sector_Ct   0x0033   200   200   140    Pre-fail  Always       -       0
  7 Seek_Error_Rate         0x000e   100   253   051    Old_age   Always       -       0
  9 Power_On_Hours          0x0032   097   097   000    Old_age   Always       -       2716
 10 Spin_Retry_Count        0x0013   100   100   051    Pre-fail  Always       -       0
 11 Calibration_Retry_Count 0x0012   100   100   051    Old_age   Always       -       0
 12 Power_Cycle_Count       0x0032   100   100   000    Old_age   Always       -       514
192 Power-Off_Retract_Count 0x0032   200   200   000    Old_age   Always       -       146
193 Load_Cycle_Count        0x0032   199   199   000    Old_age   Always       -       4787
194 Temperature_Celsius     0x0022   112   098   000    Old_age   Always       -       35
196 Reallocated_Event_Count 0x0032   200   200   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0012   200   200   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0010   100   253   000    Old_age   Offline      -       0
199 UDMA_CRC_Error_Count    0x003e   200   200   000    Old_age   Always       -       0
200 Multi_Zone_Error_Rate   0x0009   200   200   051    Pre-fail  Offline      -       0
```

Tento výpis vypadá sice na první pohled značně komplikovaně, ale jeho čtení není až tak těžké. Vlevo je název atributu, ten je ve většině případů samovysvětlující. Konkrétní hodnota daného atributu je zobrazena ve sloupci RAW_VALUE ("surová" hodnota). Surová hodnota není u některých atributů vždy zcela reprezentativní, protože disky od různých výrobců mohou zaznamenávat trošku jiná čísla (bohužel, technologie S.M.A.R.T. není standardizována, takže z části platí "každý pes, jiná ves").

Kupříkladu, výše uvedený výpis pochází z disku od firmy Western Digital. Povšimněte si, že první atribut (raw read error rate) má nulovou "surovou" hodnotu. Naopak jiný disk od firmy Seagate mi hlásí hodnotu177587740, která je nepoměrně vyšší, což ale neznačí, že by disk měl nějakou poruchu - pro disky od tohoto výrobce je to běžné a neznamená to nic znepokojivého. Podobně vysoká hodnota je u Seagatu běžná i u atributu *hardware ECC recovered* (jak je patrné z výpisu, tento disk od Western Digital tento atribut nemá). Samsungy hlásí podobně vysoké hodnoty u *hardware ECC recovered*, ale už ne u *raw read error rate*. Řada důležitých údajů má však reprezentativní a porovnatelnou hodnotu napříč disky od různých výrobců.

V některých případech se může hodit pro referenci použít interpretované hodnoty (value, worst, treshold). Interpretovaná hodnota funguje následovně: Firmware disku umí každý atribut přepočítat do reprezentativní, normalizované hodnoty. Pro každý atribut je určen práh (treshold), jehož přetečení indikuje vážný problém. Tento problém může být dvojího rázu, a to podle typu atributu - jedná-li se o atribut pre-fail, pak překročení daného prahu značí hrozící akutní selhání disku. Jedná-li se o atribut old age, pak přetečení prahu značí, že disk je již příliš starý a opotřebovaný.

Aktuální interpretovaná hodnota se nachází ve sloupci "value", ve sloupci "worst" naleznete nejhorší dosaženou hodnotu (nejblíže prahu) za celý život pevného disku, a samotný práh naleznete ve sloupci "tresh" (jako treshold - práh).

### Klíčové údaje

Mezi ty nejdůležitější údaje, na které byste se měli zaměřit, patří:

**Read error rate** indikuje počet hardwarových chyb při čtení z disku (pro některé výrobce je typická extrémně vysoká hodnota, která ovšem neznamená problém - viz výše).

**Reallocated sector count** udává počet realokovaných sektorů. Moderní pevné disky již nenechávají správu špatných sektorů na operačním systému, řeší si ho sami. Mají totiž zásobu rezervních sektorů, takže jakmile firmware pevného disku zjistí, že se zápisem do nějakého sektoru je problém, označí jej jako vadný a přesune data do rezervní oblasti. Disk se tak nadále pro operační systém chová jako disk bez špatných sektorů. Zvyšující se počet realokovaných sektorů značí problém s povrchem disku.

**Reallocation event count** je počet úspěšných i neúspěšných pokusů přesunout data do rezervní oblasti.

**Current pending sector count** označuje počet "problémových" sektorů, kde došlo k chybě čtení (takový sektor není přemapován, neboť se následné čtení ještě může podařit). Pokud se následný zápis nebo čtení tohoto sektoru podaří, tato hodnota se sníží.

**Uncorrectable sector count** je počet neopravitelných chyb při čtení nebo zápisu. Zvýšení této hodnoty indikuje problém s povrchem disku.

**Spin retry count** je počet neúspěšných pokusů o náběh disku na provozní otáčky - zvyšující se hodnota značí problém s mechanikou disku.

Další atributy jsou popsány podrobně v anglické Wikipedii, nicméně výše uvedené mají obvykle dostatečnou vypovídací schopnost.

Hodnoty jednotlivých atributů je třeba vhodně interpretovat, a k tomu vám bohužel přímé vodítko neposkytnu. Zvyšující se hodnoty klíčových atributů uvedených výše značí možný problém, tedy možné selhání disku v blízké budoucnosti, ale pouze s jistou pravděpodobností. To, že máte na disku třeba jeden realokovaný sektor, ještě nemusí vylučovat možnost, že disk bude spolehlivě fungovat ještě roky. Stejně tak fakt, že veškeré "klíčové" atributy mají nulové hodnoty, neznamená, že disk nemůže během hodiny katastrofálně selhat.

Mohu vás nicméně odkázat na studii o poruchovosti disků (PDF), kterou provedl Google, a která zohledňuje některé S.M.A.R.T. atributy a jejich vztah k pravděpodobnosti selhání disku. Tato studie je poměrně komplexní a zahrnuje i další faktory, jako je například teplota.

Tím bych tento díl ukončil. Příště se budu věnovat S.M.A.R.T. logům a jejich interpretaci v kontextu hodnot S.M.A.R.T. atributů.

## Správa linuxového serveru: S.M.A.R.T. logy

V tomto dílu budu pokračovat v popisu technologie S.M.A.R.T., konkrétně se podívám na S.M.A.R.T. logy, a na některé zajímavé závěry studie Googlu o vlivu teploty na životnost disku.

### S.M.A.R.T. logy

Jak už bylo řečeno dříve, kromě zaznamenávání řady údajů provádí technologie S.M.A.R.T. logování - pokud firmware disku detekuje chybový stav, zaznamená jej do svého vlastního vnitřního logu. Kapacita tohoto logu je bohužel velmi omezena - ukládá se pouze pět posledních záznamů. Výpis z logu může vypadat takto:

```
SMART Error Log Version: 1
ATA Error Count: 36 (device log contains only the most recent five errors)
        CR = Command Register [HEX]
        FR = Features Register [HEX]
        SC = Sector Count Register [HEX]
        SN = Sector Number Register [HEX]
        CL = Cylinder Low Register [HEX]
        CH = Cylinder High Register [HEX]
        DH = Device/Head Register [HEX]
        DC = Device Command Register [HEX]
        ER = Error register [HEX]
        ST = Status register [HEX]
Powered_Up_Time is measured from power on, and printed as
DDd+hh:mm:SS.sss where DD=days, hh=hours, mm=minutes,
SS=sec, and sss=millisec. It "wraps" after 49.710 days.

Error 36 occurred at disk power-on lifetime: 27 hours (1 days + 3 hours)
  When the command that caused the error occurred, the device was active or idle.

  After command completion occurred, registers were:
  ER ST SC SN CL CH DH
  -- -- -- -- -- -- --
  40 51 08 00 43 de e0  Error: UNC 8 sectors at LBA = 0x00de4300 = 14566144

  Commands leading to the command that caused the error were:
  CR FR SC SN CL CH DH DC   Powered_Up_Time  Command/Feature_Name
  -- -- -- -- -- -- -- --  ----------------  --------------------
  c8 00 08 00 43 de e0 08      00:28:29.500  READ DMA
  27 00 00 00 00 00 e0 08      00:28:29.500  READ NATIVE MAX ADDRESS EXT
  ec 00 00 00 00 00 a0 0a      00:28:29.500  IDENTIFY DEVICE
  ef 03 42 00 00 00 a0 0a      00:28:29.400  SET FEATURES [Set transfer mode]
  27 00 00 00 00 00 e0 08      00:28:29.400  READ NATIVE MAX ADDRESS EXT
```

Povšimněte si údajů o celkovém počtu chyb (druhý řádek výpisu). Samotná chyba je popsána poměrně detailně, včetně série příkazů, které vedly k chybovému stavu, a časového údaje, ze kterého je možné v tomto případě vyčíst, že k poslední chybě (chybě č. 36) došlo ve 27. hodině fungování pevného disku. S.M.A.R.T. logy je nicméně nutné brát v kontextu k ostatním údajům - teprve pak mají vhodnou vypovídací schopnost. Časovou značku chyby (27 hodin) lze porovnat se S.M.A.R.T. údajem "power on hours", který udává celkovou dobu fungování disku:

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  9 Power_On_Hours          0x0012   100   100   000    Old_age   Always       -       4339
```

Z toho je vidět, že chyba nastala velmi brzy po zařazení disku do užívání, a od té doby nenastal žádný problém. Což je povzbudivé. Podívejte se ještě na ostatní důležité údaje:

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000b   100   100   016    Pre-fail  Always       -       0
  4 Start_Stop_Count        0x0012   100   100   000    Old_age   Always       -       72
  5 Reallocated_Sector_Ct   0x0033   100   100   005    Pre-fail  Always       -       0
  7 Seek_Error_Rate         0x000b   100   100   067    Pre-fail  Always       -       0
  9 Power_On_Hours          0x0012   100   100   000    Old_age   Always       -       4339
 10 Spin_Retry_Count        0x0013   100   100   060    Pre-fail  Always       -       0
193 Load_Cycle_Count        0x0012   100   100   000    Old_age   Always       -       92
194 Temperature_Celsius     0x0002   171   171   000    Old_age   Always       -       35 (Lifetime Min/Max 21/49)
196 Reallocated_Event_Count 0x0032   100   100   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0022   100   100   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0008   100   100   000    Old_age   Offline      -       0
199 UDMA_CRC_Error_Count    0x000a   200   200   000    Old_age   Always       -       0
```

Realokovaných sektorů je nula, reallocated event count je nulový, raw read error rate také (jedná se o disk Hitachi, který u tohoto atributu udává reprezentativní surovou hodnotu), offline uncorrectable také vykazuje nulovou hodnotu. Z toho je patrné, že disk by měl být v pořádku, přestože má v logu spoustu chyb.

Právě na tomto příkladu bych rád ukázal a zdůraznil, že chybové hlášky v logu ještě nutně nemusí znamenat problém s diskem. V tomto konkrétním případě došlo k přerušení napájení, když disk prováděl zápis. Výsledkem bylo několik sektorů u kterých došlo k nekonzistenci mezi daty a ECC záznamy (Error Checking and Correction). Z toho disk při čtení usoudil, že sektor nelze přečíst (jakmile porovnal data a příslušné ECC záznamy), tudíž došlo k neopravitelné chybě čtení (viz UNC = uncorrectable error in data v chybové hlášce v logu). Po přepsání těchto sektorů problém zmizel a disk nyní funguje bezvadně.

Pokud tedy budete posuzovat zdraví nějakého pevného disku, a v logu objevíte nějaké chybové hlášky, pak je rozhodně vnímejte v kontextu se S.M.A.R.T. údaji (a pokud jste na pochybách, pak disk otestujte).

### Teplota a zdraví pevných disků

Pokud se podíváte na studii Googlu (PDF), na kterou jsem vás odkazoval v minulém díle, pak zjistíte, že teplota sice hraje roli v rámci životnosti disku, ale možná ne tak, jak byste očekávali. Z výsledků této studie totiž mj. vyplývá, že diskům vadí spíše příliš nízké teploty (kolem 15°C) než příliš vysoké. Teplota do výše 50°C ovlivňuje negativně životnost disku méně než teploty pod 25°C. Teplotní optimum se podle dané studie nachází někde kolem 40°C.

Některé disky umí v rámci S.M.A.R.T. zaznamenat nejen svou aktuální teplotu, ale i celoživotní minimum a maximum:

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
194 Temperature_Celsius     0x0002   171   171   000    Old_age   Always       -       35 (Lifetime Min/Max 21/49)
```

Přirozeně, nejpodstatnější pro pevný disk je spíše průměrná teplota než minimum a maximum.

### S.M.A.R.T. a vnější chyby

Růst hodnot některých S.M.A.R.T. atributů nemusí být nutně spojen s opotřebením nebo blížící se poruchou pevného disku. Některé hodnoty mohou růst třeba následkem vadné kabeláže (typicky UDMA CRC Error Count), apod. - tyto faktory je nutné brát v potaz a pokusit se jejich vliv při diagnostice zohlednit.

### S.M.A.R.T. a BIOS

BIOSy moderních základních desek obvykle umožňují při bootování hlídat stav S.M.A.R.T. a hlásit případný problém. Na tuto funkcionalitu není dobré spoléhat - hlídá totiž pouze jednu jedinou hodnotu, a tou je celkový stav. Ve výpisu nástroje smartctl je tato hodnota také k dispozici:

```
=== START OF READ SMART DATA SECTION ===
```

SMART overall-health self-assessment test result: PASSED

Co tato hodnota znamená? V podstatě pouze to, že nebyl překročen práh (treshold) žádného ze S.M.A.R.T. atributů. To je samozřejmě důležitá hodnota a ve chvíli, kdy bude udávat "FAILED", je vhodné data zálohovat a disk co nejdříve vyřadit. Naneštěstí to, že tato hodnota udává "PASSED", ještě neznamená, že nelze bližším studiem konkrétních hodnot S.M.A.R.T. atributů odhadnout blížící se potenciální problém dlouho před tím, než se tato hodnota změní.

**S.M.A.R.T. a firmware disku**

S.M.A.R.T. je de facto software, který je součástí firmwaru disku. A jako každý software, i implementace S.M.A.R.T. (potažmo firmware disku jako takový) může mít chyby. Takové chyby občas postihnou některé série pevných disků, a stává se to čas od času snad úplně každému výrobci pevného disku.

Disk tedy může selhat nebo vypadnout z diskového pole nejenom následkem mechanického poškození nebo opotřebení, ale i následkem chyby ve firmwaru, která může provést téměř cokoliv počínaje vracením nepravdivých S.M.A.R.T. údajů a úplným zamrznutím disku konče. I to je jeden z důvodů, proč se do diskových polí doporučuje dávat různé disky od různých výrobců.

Tímto bych pro tentokrát skončil, s tím, že příště se podívám na S.M.A.R.T. testy a na to, jak lze změny S.M.A.R.T. údajů monitorovat.

## Správa linuxového serveru: S.M.A.R.T. testy a monitorování

V tomto díle završím problematiku S.M.A.R.T. a zdraví pevných disků. Podívám se na využití S.M.A.R.T testů a prozradím vám, jak lze S.M.A.R.T. údaje monitorovat.

### S.M.A.R.T. testy

Třetí a poslední funkcionalitou, kterou S.M.A.R.T. poskytuje, je testování pevného disku (tedy jakási samodiagnostika). Tyto testy probíhají na pozadí, lze je tedy použít i na běžícím systému. Je jasné, že intenzivní práce s diskem průběh příslušného testu pozdrží.

Testy jsou celkem tři, krátký (short), dlouhý (long) a "přepravní" (conveyance). Každý z těchto testů dělá něco malinko jiného, krátký testuje základní funkčnost disku, dlouhý provádí totéž, ale trošku důkladněji (a kromě toho testuje i povrch disku) a conveyance test je schopen určit poškození disku způsobené přepravou (ne všechny pevné disky jej ale umí provést).

Délka testů se liší model od modelu, přičemž očekávanou délku testu se dozvíte už z celkového výpisu (smartctl -a /dev/disk):

```
Short self-test routine
recommended polling time:        (   1) minutes.
Extended self-test routine
recommended polling time:        ( 235) minutes.
```

Testy je možné zadávat a vyhodnocovat opět pomocí nástroje smartctl a přepínače -t. Krátký test se spustí příkazem:

```
smartctl -t short /dev/sda
```

U dlouhého testu se u parametru -t použije hodnota long, a u přepravního testu se použije hodnotaconveyance. Pokud test tímto způsobem zadáte, objeví se vám podobný výpis:

```
debian ~# smartctl -t short /dev/sda
smartctl version 5.38 [x86_64-unknown-linux-gnu] Copyright (C) 2002-8 Bruce Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
Sending command: "Execute SMART Short self-test routine immediately in off-line mode".
Drive command "Execute SMART Short self-test routine immediately in off-line mode" successful.
Testing has begun.
Please wait 2 minutes for test to complete.
Test will complete after Tue Jan 12 15:47:26 2010

Use smartctl -X to abort test.
```

Z výpisu se dozvíte očekávanou dobu trvání testu i očekávaný čas jeho dokončení. Nástroj smartctl vám také radí, že test lze zrušit pomocí parametru -X.

Výsledek testu se dozvíte opět z výpisu smartctl -a /dev/disk:

```
SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Extended offline    Completed without error       00%      2290         -
# 2  Extended offline    Completed without error       00%       302         -
# 3  Short offline       Completed without error       00%        83         -
# 4  Extended offline    Completed without error       00%        70         -
# 5  Extended offline    Interrupted (host reset)      40%        65         -
# 6  Extended offline    Completed: read failure       90%        27         14566144
# 7  Extended offline    Completed: read failure       90%        26         14566144
```

Zde vidíte celkem sedm testů, jsou seřazeny inverzně dle doby, kdy byly provedeny. Všechny typy testů kromě třetího jsou "dlouhé" (long), čtyři naposledy prováděné testy (1-4) nevykazují žádný problém, pátý test byl přerušen restartem počítače zhruba v 60% průběhu, přičemž šestý a sedmý skončil s chybou. U těchto dvou testů vidíte i číslo bloku (14566144), kde k chybě došlo.

Tento výpis náleží disku Hitachi, kterého se týkal onen problém s výpadkem napájení a následné nekonzistenci mezi ECC záznamem a daty v osmi sektorech, která vyvolala chybu čtení. Srovnejme to nyní s výpisem ze S.M.A.R.T. logu:

```
Error 36 occurred at disk power-on lifetime: 27 hours (1 days + 3 hours)
When the command that caused the error occurred, the device was active or idle.

After command completion occurred, registers were:
ER ST SC SN CL CH DH
-- -- -- -- -- -- --
40 51 08 00 43 de e0  Error: UNC 8 sectors at LBA = 0x00de4300 = 14566144

...
```

Jak je vidět, číslo bloku se shoduje (14566144). Z výpisu testů je jasně patrné, že tato chyba byla odstraněna (pozdější testy proběhly bez chyby) a nulová hodnota S.M.A.R.T. atributu "reallocated sectors count" potvrzuje, že se nejednalo o špatné sektory, které by bylo nutné přemapovat. Tento disk je tedy zcela v pořádku.

### S.M.A.R.T. monitorování

Hodnoty S.M.A.R.T. atributů je možné sledovat prostřednictvím démona, který bude zachytávat změny hodnot a ve výchozí konfiguraci zapisovat změny do systémového logu. Tento démon je součástí balíčkusmartmontools, tudíž pokud máte nainstalován ten, stačí démona jen povolit.

V Debianu je za tímto účelem třeba upravit konfigurační soubor /etc/default/smartmontools a odkomentovat následující řádku:

```
start_smartd=yes
```

Démona je možné konfigurovat i poněkud precizněji, a to prostřednictvím konfiguračního souboru/etc/smartd.conf. V něm je možné nastavit mj. kupříkladu zasílání varování o změnách na zadaný e-mail.

### Parkování hlaviček a úsporný režim disků

Jedním z atributů, jehož hodnoty S.M.A.R.T. sleduje, je i počet zaparkování hlaviček disku (load cycle count) za celou dobu jeho života. Toto množství je totiž svým způsobem omezené - výrobci disků udávají vždy pouze určitý počet parkování, které by disk měl zvládnout. Desktopové a serverové disky obvykle zvládnou podstatně méně než disky určené pro notebooky (kde je parkování vítaným prostředkem jednak pro ochranu pevného disku před poškozením daným manipulací s notebookem, a jednak jako součást procesu řízení spotřeby).

Pevné disky mají obvykle vlastní systém řízení spotřeby (power management), který lze u řady disků ovlivňovat parametrem -B nástroje hdparm, jehož hodnoty jdou od 1 do 255 s tím, že u jedničky dochází k maximálnímu šetření energie, zatímco u 254 by měl být maximálně zohledněn výkon (na úkor spotřeby). Hodnota 255 by měla řízení spotřeby vypnout (ne všechny disky ji bohužel podporují). Hodnoty od 1 do 127 povolují zastavení disku (spindown), hodnoty od 128 výše nikoliv. Nastavení hodnoty režimu řízení spotřeby by se u konkrétního pevného disku provedlo příkazem:

```
hdparm -B 128 /dev/disk
```

Pokud byste chtěli disku zabránit v parkování hlaviček a parametr -B by nepomohl, můžete ještě disku nařídit, ať neprovádí zastavení disku:

```
hdparm -S 0 /dev/disk
```

V případě domácích serverů byste asi řekli, že se jich tato problematika netýká. To jsem si myslel také, dokud jsem nenarazil na jednu sérii úsporných disků jisté značky, u kterých docházelo k příliš častému parkování hlaviček, a aby to nebylo tak jednoduché, příslušné disky nereagovaly na adekvátní příkazyhdparm -B a hdparm -S. Jediné, co zbývalo, byla úprava firmwaru (já jsem se do ní nepouštěl a disk raději rovnou vyměnil).

Bohužel, snaha o úsporu energie, související marketing a možná laxní přístup některých výrobců k testování svých pevných disků pod více operačními systémy, může vést k podobným nepříjemným situacím.

Z tohoto důvodu nebývá úplně od věci kontrolovat u pevných disků i počet parkování hlaviček (load cycle count):

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
193 Load_Cycle_Count        0x0032   199   199   000    Old_age   Always       -       4789
```

Pevné disky obvykle snesou hodnoty v řádu desítek až set tisíců, u notebookových disků tato hodnota může být i 500000. Záleží na konkrétním modelu. Příliš rychle rostoucí hodnotu load cycle count je vhodné řešit způsobem naznačeným výše. Debian nastavuje hodnoty hdparm při bootu, a tyto hodnoty lze upravovat i pro jednotlivé disky v konfiguračním souboru /etc/hdparm.conf.

**Vývoj pravděpodobnosti selhání disku v čase**

Statistická pravděpodobnost selhání pevného disku není po dobu jeho existence stejná. Vzhledem k přepravě a výrobnímu procesu je zvýšená pravděpodobnost selhání disku během prvních měsíců jeho užívání (proto se také vyplácí provádět zátěžové testy). Poté pravděpodobnost selhání klesne, a s rostoucím časem opět roste (disk stárne).

Z toho vyplývá jediné - příliš mladému a příliš starému disku byste neměli bezvýhradně věřit. I proto se vyplatí diskové pole obměňovat postupně v delším rozmezí, spíše než naráz. Kandidáty na vyloučení vybírejte s ohledem na S.M.A.R.T. údaje (disky s realokovanými sektory nebo s nenulovou hodnotou u některých kritických atributů by měly být vyměněny v první řadě).

**Jak pomoci S.M.A.R.T.u, když si nejste jisti**

Pokud si u pevného disku nejste jisti jeho zdravím, a nemáte na něm důležitá data, můžete S.M.A.R.T.u trošku pomoci použitím programu badblocks a jeho read/write testu. Ten můžete spustit příkazem:

```
badblocks -s -w /dev/disk
```

Pozor - tímto příkazem přepíšete všechna data na daném pevném disku! Pokud si nemůžete dovolit data přesunout, můžete použít dlouhý S.M.A.R.T. test, který povrch disku prověří.

Tím bych problematiku S.M.A.R.T. uzavřel. Příště se budu věnovat praktické stránce LVM, a poté dm-crypt/LUKS.

## Správa linuxového serveru: LVM prakticky

LVM byl v tomto seriálu představen na teoretické úrovni v jednom z prvních dílů. Nyní převedu teorii do praxe a ukážu vám, jak se s LVM pracuje. V tomto dílu proberu základní operace s LVM.

Abyste se v následujícím článku neztratili, doporučujeme si nejdříve přečíst předchozí díly seriálu, především však článek Správa linuxového serveru: LVM a diskové šifrování.

### Vytvoření LVM svazku

Základem pro LVM jsou fyzické svazky (physical volumes). Nejprve tedy musíte vytvořit fyzické svazky, které později začleníte do skupiny svazků (volume group), kterou pak rozdělíte na jednotlivé logické svazky (logical volumes).

Ukážu vám to na příkladu. Řekněme, že máte v počítači tři pevné disky, na každém jeden oddíl vyplňující celý disk:

```
debian:~# cat /proc/partitions
major minor  #blocks  name

   3     0    5242880 hda
   3     1    5237158 hda1
   3    64    5242880 hdb
   3    65    5237158 hdb1
  22    64    5242880 hdd
  22    65    5237158 hdd1
```

Prvním krokem je vytvoření fyzického svazku. Vytvoříte tedy fyzický svazek z oddílu hdb1:

```
debian:~# pvcreate /dev/hdb1
Physical volume "/dev/hdb1" successfully created
debian:~#
```

Tímto byl vytvořen fyzický svazek. Podrobné informace o fyzickém svazku, který byl právě vytvořen, lze zobrazit s použitím nástroje pvdisplay, který vypíše informace o konkrétním fyzickém svazku, pokud mu jako parametr zadáte nějaký konkrétní, nebo vypíše informace o všech fyzických svazcích, o kterých ví (v tomto případě pouze o tom jednom), pokud jej spustíte bez parametrů:

```
debian:~# pvdisplay
"/dev/hdb1" is a new physical volume of "4.99 GB"
--- NEW Physical volume ---
PV Name               /dev/hdb1
VG Name
PV Size               4.99 GB
Allocatable           NO
PE Size (KByte)       0
Total PE              0
Free PE               0
Allocated PE          0
PV UUID               3wvOQk-M0uS-XJN6-nlrl-x8Uz-CJDF-in2fEs
```

Velikost svazku je 4.99GB, většina ostatních hodnot je poznamenána tím, že fyzický svazek není součástí žádné skupiny svazků (volume group). Abyste mohli fyzický svazek využít, musíte jej začlenit do skupiny svazků. Jelikož v tuto chvíli není k dispozici žádná skupina svazků, vytvoříte novou, třeba s názvem "data", s pomocí právě tohoto fyzického svazku:

```
debian:~# vgcreate data /dev/hdb1
Volume group "data" successfully created
debian:~#
```

Pro zjišťování informací o skupině svazků je k dispozici podobný nástroj - vgdisplay, jehož výpis v tuto chvíli vypadá takto:

```
debian:~# vgdisplay
--- Volume group ---
VG Name               data
System ID
Format                lvm2
Metadata Areas        1
Metadata Sequence No  1
VG Access             read/write
VG Status             resizable
MAX LV                0
Cur LV                0
Open LV               0
Max PV                0
Cur PV                1
Act PV                1
VG Size               4.99 GB
PE Size               4.00 MB
Total PE              1278
Alloc PE / Size       0 / 0
Free  PE / Size       1278 / 4.99 GB
VG UUID               SmBDKH-iTL3-K3aM-I4e6-qgyZ-XL05-wxG7P1

debian:~#
```

Z výpisu je vidět použitý formát (lvm2), počet fyzických svazků přiřazených do skupiny, dále celkovou velikost skupiny svazků i její obsazení. Všimněte si také zkratky PE, jejíž význam odhaluje jádro fungování LVM.

LVM pracuje na fyzické úrovni tak, že si rozdělí fyzické svazky do bloků určité velikosti, které pak přiděluje jednotlivým logickým svazkům. Tyto bloky jsou nazývány jako "physical extent" (dále PE). Ve výpisu je vidět, že jeden PE má v tomto případě velikost 4MB a skupina svazků jich má k dispozici celkem 1278, z nichž žádný není dosud alokován (přiřazen logickému svazku).

Ještě než se vrátím k vytváření logického svazku, upozorním ještě na to, čeho si pozorný čtenář jistě všiml, a sice nápadné podobnosti jednotlivých příkazů pro správu LVM, které se liší pouze úvodními dvěma písmeny. V případě LVM je dobré si v paměti uchovat anglické názvy všech pojmů jako "physical volume", "volume group" a "logical volume", protože počáteční písmena daných názvů tvoří prefix pro sadu příkazů, kterými je lze spravovat. Tudíž, pokud operujete s fyzickými svazky, použijete prefix pv následovaný slovíčky jako display, create, extend, atd., a analogický postup použijete v případě ostatních komponent LVM.

Avšak zpět k vytváření LVM. V tuto chvíli je tedy vytvořena skupina svazků z jednoho fyzického svazku. Zbývá poslední krok, a sice vytvoření logického svazku:

```
debian:~# lvcreate -L 4.99G data -n zalohy
Rounding up size to full physical extent 4.99 GB
Logical volume "zalohy" created
```

Parametr -L udává velikost logického svazku, v tomto případě 4.99G, kde G značí gigabyty. Parametr -numožňuje dát příslušnému logickému svazku jméno, v tomto případě "zalohy".

Příslušný logický svazek je k dispozici jako blokové zařízení /dev/mapper/data-zalohy a lze s ním zacházet jako s kterýmkoliv diskovým oddílem (nebo blokovým zařízením obecně). Příslušné zařízení má i symbolický odkaz /dev/data/zalohy, tudíž je k němu možné přistupovat i takto, o něco málo pohodlněji.

Pokud byste nyní použili nástroj lvdisplay k zobrazení informací o nově vytvořeném logickém svazku, dostali byste výpis podobný tomuto:

```
debian:~# lvdisplay
  --- Logical volume ---
  LV Name                /dev/data/zalohy
  VG Name                data
  LV UUID                hf1rYC-1ECA-PopX-AVwx-qquW-jOHI-EOnWDl
  LV Write Access        read/write
  LV Status              available
  # open                 0
  LV Size                4.99 GB
  Current LE             1278
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:0
```

Na příslušném logickém svazku nyní zbývá už jen vytvořit souborový systém, v tomto případě ext3:

```
mkfs.ext3 /dev/data/zalohy
```

Souborový systém na novém logickém svazku lze nyní připojit a začít na něj kopírovat data.

## Rozšíření logického svazku

Úžasnou výhodou LVM je kromě nezávislosti logického členění svazků na fyzických svazcích také možnost provádět různé změny, a to klidně i za běhu systému. Mezi tu nejjednodušší změnu patří zvětšení logického svazku, kterou nyní předvedu.

Podívejte se nejprve na to, co vypíše vgdisplay o skupině svazků (z výpisu jsem vybral pouze relevantní řádky):

```
debian:~# vgdisplay
  --- Volume group ---
  VG Name                data
  ...
  Total PE               1278
  Alloc PE / Size        1278 / 4.99 GB
  Free  PE / Size        0 / 0
  ...
```

Jak je vidět, všechny PE (physical extents) byly již alokovány, to znamená, že v tuto chvíli již není z čeho vytvořit nový logický svazek. Skupina svazků se ale může sestávat z více než jednoho fyzického svazku. Je tedy možné přidat další blokové zařízení (ať už disk, oddíl, diskové pole, apod.) a rozšířit dostupnou kapacitu skupiny svazků:

```
debian:~# pvcreate /dev/hdd1
Physical volume "/dev/hdd1" successfully created
debian:~# vgextend data /dev/hdd1
Volume group "data" successfully extended
```

Prvním příkazem byl z diskového oddílu hdd1 vytvořen fyzický svazek, ve druhém kroku byla rozšířena skupina svazků "data" o nový fyzický svazek. Nástroj vgdisplay nyní potvrzuje navýšení kapacity skupiny svazků:

```
  ...
  VG Size                9.98 GB
  PE Size                4.00 MB
  Total PE               2556
  Alloc PE / Size        1278 / 4.99 GB
  Free  PE / Size        1278 / 4.99 GB
  ...
```

V tuto chvíli již tedy je z čeho rozšířit stávající logický svazek "zalohy". Jak rozšíření skupiny svazků, tak rozšíření logického svazku je možné provádět za běhu systému. Není ani potřeba odpojit souborový systém na daném zařízení:

```
debian:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/data-zalohy  5.0G  374M  4.3G   8% /mnt
debian:~# lvextend -L +2G /dev/data/zalohy
  Extending logical volume zalohy to 6.99 GB
  Logical volume zalohy successfully resized
debian:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/data-zalohy  5.0G  374M  4.3G   8% /mnt
```

Výstup příkazu df jsem uvedl pro lepší pochopení toho, co se stalo. Jak je patrné, ačkoliv byl logický svazek zvětšen o 2GB, tedy na 7GB celkem, souborový systém stále hlásí kapacitu 5GB. Logický svazek se v tomto směru chová jako diskový oddíl, který byl rozšířen, ale souborový systém na něm nebyl nijak pozměněn, tudíž hlásí stejnou kapacitu jako před tím. Aby mohl souborový systém nově vzniklý extra prostor využít, je třeba jej rozšířit ručně.

V případě souborového systému ext3 lze použít nástroje resize2fs, který umí změnit jeho velikost a roztáhnout jej na celý oddíl. Tuto změnu je o něco lepší provádět na odpojeném souborovém systému, jelikož se jedná o mírně riskantní operaci, ale lze ji samozřejmě provést i na připojeném souborovém systému přímo za běhu:

```
debian:~# resize2fs /dev/data/zalohy
resize2fs 1.41.3 (12-Oct-2008)
Filesystem at /dev/data/zalohy is mounted on /mnt; on-line resizing required
old desc_blocks = 1, new_desc_blocks = 1
Performing an on-line resize of /dev/data/zalohy to 1832960 (4k) blocks.
The filesystem on /dev/data/zalohy is now 1832960 blocks long.

debian:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/data-zalohy  6.9G  375M  6.2G   6% /mnt
```

Nyní již souborový systém vykazuje správnou velikost, a tím byl proces rozšíření skupiny svazků i logického svazku dokončen.

## Zmenšení logického svazku

Redukovat velikost souborového systému i logického svazku možné je, ale v tomto případě se již patrně nevyhnete nutnosti daný souborový systém odpojit, alespoň v případě ext3.

Postup je třeba provést ve správném pořadí (špatné pořadí povede skoro jistě ke ztrátě dat) - nejprve je třeba změnit velikost souborového systému, a teprve poté lze bezpečně změnit velikost logického svazku. Nástroj resize2fs se naštěstí chová opatrně a nepředpokládá, že uživatel vždy ví, co dělá:

```
debian:~# resize2fs /dev/data/zalohy 6G
resize2fs 1.41.3 (12-Oct-2008)
Filesystem at /dev/data/zalohy is mounted on /mnt; on-line resizing required
On-line shrinking from 1832960 to 1572864 not supported.
```

Při pokusu zmenšit připojený souborový systém začal resize2fs protestovat. Odpojíte tedy příslušný souborový systém a zkusíte to znovu:

```
debian:~# umount /dev/data/zalohy
debian:~# resize2fs /dev/data/zalohy 6G
resize2fs 1.41.3 (12-Oct-2008)
Please run 'e2fsck -f /dev/data/zalohy' first.
```

V tuto chvíli resize2fs zjistil, že souborový systém není označen jako "čistý" (patrně kvůli dřívějšímu zvětšení), tudíž doporučuje nechat souborový systém prověřit nástrojem fsck:

```
debian:~# e2fsck -f /dev/data/zalohy
e2fsck 1.41.3 (12-Oct-2008)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

/dev/data/zalohy: ***** FILE SYSTEM WAS MODIFIED *****
/dev/data/zalohy: 16269/458752 files (0.1% non-contiguous), 124768/1832960 blocks
```

Je vidět, že skutečně k nějaké úpravě, respektive opravě souborového systému došlo. Následuje třetí a poslední pokus o zmenšení souborového systému:

```
debian:~# resize2fs /dev/data/zalohy 6G resize2fs 1.41.3 (12-Oct-2008) Resizing the filesystem on /dev/data/zalohy to 1572864 (4k) blocks. The filesystem on /dev/data/zalohy is now 1572864 blocks long.
```

Operace proběhla úspěšně a souborový systém je nyní možné opět připojit. Po opětovném připojení lze ověřit, že byl opravdu zmenšen:

```
debian:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/data-zalohy   6.0G  374M  5.3G   7% /mnt
```

Úplně posledním krokem celého procesu je zmenšení samotného logického svazku:

```
debian:~# lvreduce -L 6G /dev/data/zalohy
WARNING: Reducing active and open logical volume to 6.00 GB
THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce zalohy? [y/n]: y
Reducing logical volume zalohy to 6.00 GB
Logical volume zalohy successfully resized
```

Zde je třeba si dát velký pozor na to, aby velikost logického svazku nebyla nikdy menší než velikost souborového systému. Varování programu lvreduce je v tomto ohledu dobré brát velmi vážně.

Tím bych pro dnešek skončil. Příště se podívám na to, jak za běhu data přesouvat mezi fyzickými svazky a proberu také práci se snapshoty.

## Správa linuxového serveru: LVM a snapshoty

Minule jsem nakousl LVM a probral základní operace s ním. Dnes se podívám na přesun dat za běhu systému a tvorbu a práci se snapshoty.

### Přesun dat za běhu systému

Předpokládejme, že byl do systému přidán nový, velký pevný disk s tím, že starých dvou menších disků se chcete zbavit. Abyste to mohli udělat, musíte přenést data z dvou menších disků na velký, a následně odebrat příslušné fyzické svazky. K tomu slouží nástroj pvmove, který umí přesouvat data mezi jednotlivými fyzickými svazky, a to přímo za provozu (vzhledem k tomu, že vrstva logických svazků je v rámci LVM oddělena od fyzických svazků, nepředstavuje tato procedura pro souborové systémy na daných logických svazcích žádnou změnu).

Pokud zadáte nástroji pvmove pouze jeden fyzický svazek, přesune všechna data z daného svazku na ostatní fyzické svazky v dané skupině svazků:

```
debian:~# pvmove /dev/hdb1
/dev/hdb1: Moved: 37.2%
/dev/hdb1: Moved: 67.3%
/dev/hdb1: Moved: 100.0%
```

V tuto chvíli je fyzický svazek hdb1 volný, a je tedy možné jej odstranit ze skupiny svazků:

```
debian:~# vgreduce data /dev/hdb1
```

Teprve teď je možné zavolat pvremove k odstranění logického svazku, nebo, přesněji řečeno, příslušných metadat, které si na daný oddíl LVM zapsalo:

```
debian:~# pvremove /dev/hdb1
```

Tento krok je samozřejmě nepovinný a není nutný - zejména, pokud máte v plánu přiřadit dané fyzické svazky jiné skupině svazků. Tentýž postup lze provést i s druhým diskem:

```
debian:~# pvmove /dev/hdd1
/dev/hdd1: Moved: 32.2%
/dev/hdd1: Moved: 64.0%
/dev/hdd1: Moved: 97.7%
/dev/hdd1: Moved: 100.0%
debian:~# vgreduce data /dev/hdd1
Removed "/dev/hdd1" from volume group "data"
debian:~# pvremove /dev/hdd1
Labels on physical volume "/dev/hdd1" successfully wiped
```

V tuto chvíli jsou všechna data přesunuta ze dvou menších disků na jeden větší. V případě výpadku napájení nebo pádu systému je po opětovném nastartování možné v přesunu pokračovat tím, že znovu spustíte nástroj pvmove, bez parametrů. Proceduru je možné i přerušit, použijete-li parametr --abort.

Stav fyzických svazků je možné rychle prověřit pomocí pvs (podobnost s ls není čistě náhodná):

```
debian:~# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/hdc1  data lvm2 a-   10.99G 4.99G
```

Podobným způsobem můžete vyvolat informace o logických svazcích (nástroj lvs) nebo skupinách svazků (nástroj vgs). Tyto nástroje poskytují v porovnání s nástroji *display pouze základní přehled.

Pokud by se vám někdy stalo, že zapomenete provést vgreduce a odstraníte prázdný, ale do skupiny zařazený fyzický svazek (navzdory protestům nástroje pvremove, kterému je v takovém případě nutné ještě přidat parametr -ff, aby tuto nebezpečnou operaci provedl), dojde k poměrně vážnému chybovému stavu, zejména v případě starších verzí LVM. Následky této nepříjemné situace je naštěstí možné jednoduše odstranit, a to následujícím příkazem:

```
vgreduce --removemissing volume_group
```

Na této hypotetické situaci bych rád ukázal tři věci. Jednak to, že varování nástrojů pro správu LVM je dobré nebrat na lehkou váhu a spíše se nejprve důkladně rozmyslet, jestli danou věc děláte správně, a teprve pak se pokusit obejít nějaký bezpečnostní mechanismus.

Za druhé, nástroje pro správu LVM umí ledacos, včetně postupů pro zotavení z nějakého problémového stavu. Studiem manuálových stránek se pak můžete vyhnout časově náročným a velice komplikovanýmručním postupům.

A konečně za třetí - LVM představuje další funkční vrstvu mezi médiem a souborovým systémem, jejíž případné chyby (nebo chyby administrátora při její správě) mohou negativně ovlivnit vaše data a zkomplikovat vám práci při případných záchranných operacích, pokud třeba selže HW. V této souvislosti opět zopakuji, že zálohování je nutnost.

### Snapshoty

Co je snapshot? Fotografové by tento termín přeložili asi jako "momentka". V případě LVM se jedná o "momentku" daného logického svazku a všech dat na něm, která se v daném okamžiku jakoby zmrazí. K této "momentce" pak můžete přistupovat jako k jinému logickému svazku (nebo obecně jako k blokovému zařízení), zatímco původní logický svazek byl již modifikován.

Snapshoty lze provádět i za běhu s připojeným souborovým systémem, i když tento postup může vést k nekonzistentním datům. Je to dáno tím, že snapshot je prováděn na úrovni "pod" souborovým systémem, o jehož stavu nemá LVM žádné informace (a naopak, souborový systém netuší, že ve funkční vrstvě pod ním došlo k nějakému zmražení stavu). Pokud tedy v okamžiku vytvoření snapshotu nějaká aplikace právě provádí zápis dat do souboru, nebo si data k zapsání na disk teprve uchovává v paměti s cílem je později uložit, je jasné, že snapshot bude obsahovat nekonzistentní data.

Souborový systém je možné před pořízením snapshotu odpojit nebo přepojit do režimu read-only, což by se, jen tak pro zajímavost, provedlo takto:

```
mount -o remount,ro /dev/zarizeni
```

LVM2, tedy aktuální implementace LVM v Linuxu, umí se snapshoty pracovat i v režimu pro zápis. Tím se liší od původní implementace LVM verze 1. LVM v Linuxu je schopné zahodit snapshot a nově i reintegrovat snapshot do původního svazku (merge). Tato poslední možnost však není k dispozici v Debianu Lenny, jelikož ten používá starší jádro i LVM nástroje. Dokonce jsem měl problémy s touto funkcionalitou i na Arch Linuxu s téměř aktuálním jádrem 2.6.32, pomohla až kompilace nového jádra 2.6.33, které se ještě ani nedostalo do repozitářů. Reintegrace snapshotů je tedy v tuto chvíli z hlediska serverů a jejich správy spíše hudbou budoucnosti, alespoň v případě "stabilních" distribucí jako Debian.

K čemu se dá využít snapshotů? K mnohému. Lze si díky nim vytvořit pojistku před nějakou radikálnější změnou (třeba upgrade celého systému na novou verzi), kterou je pak možné vrátit zpět, pokud se operace nezdaří nebo něco přestane fungovat.

S pomocí snapshotů lze usnadnit proces zálohování, i když se jistému krátkému výpadku služeb nevyhnete, chcete-li mít zálohu konzistentní. Abyste zajistili konzistenci dat, postačí na okamžik zastavit běžící démony přistupující k danému svazku, poté jej přepojit do režimu pouze pro čtení (nebo na chvilku odpojit), vytvořit snapshot, souborový systém znovu připojit (nebo obnovit možnost zápisu) a vypnuté démony znovu spustit. Poté je možné snapshot připojit a z něj provést zálohu, jejíž pořízení může trvat jakkoliv dlouho. Výsledkem je jen minimální výpadek služeb v porovnání s konzistentní a korektně provedenou zálohou bez snapshotů.

Ale dost již teorie, dovolte mi nyní práci se snapshoty demonstrovat prakticky. Předpokládejme, že máte k dispozici testovací logický svazek snaptest ve skupině svazků vg. Testovacímu svazku přiřadíte 5GB:

```
[root@arch ~]# lvcreate -L 5G -n snaptest vg
  Logical volume "snaptest" created
```

Na svazku vytvoříte souborový systém, připojíte jej a překopírujete na něj nějaká data. Pro lepší představu uvádím příklad výpisu adresáře na daném logickém svazku před vytvořením snapshotu:

```
drwxr-xr-x   7 root root  4096 Feb 25 19:15 cache
drwxr-xr-x 227 root root 12288 Feb 25 19:15 doc
drwxr-xr-x   2 root root  4096 Feb 25 18:53 obsolete
```

V tomto stavu se rozhodnete vytvořit snapshot. Snapshot se vytváří úplně stejně jako logický svazek, pouze se přidá parametr -s:

```
[root@arch ~]# lvcreate -s -L 2G -n momentka vg/snaptest
  Logical volume "momentka" created
```

Všimněte si, že je nutné snapshotu přiřadit nějaký prostor (v tomto případě 2GB), kam se pak zapisují změny oproti fyzickému svazku, ze kterého je vytvořen.
V tuto chvíli začnete provádět změny. Na fyzickém svazku odstraníte adresář obsolete obsahující stará a nezajímavá data. Přidáte nový adresář archive, kam umístíte nějaké zálohy. Výsledek bude vypadat takto:

```
drwxr-xr-x   5 root root  4096 Feb 25 20:53 archive
drwxr-xr-x   7 root root  4096 Feb 25 19:15 cache
drwxr-xr-x 227 root root 12288 Feb 25 19:15 doc
```

V tuto chvíli máte na výběr mezi dvěma variantami. Můžete snapshot zrušit, nebo jej reintegrovat do původního logického svazku (merge).

***Zrušení snapshotu***

Dojde-li ke zrušení snapshotu, přijdete o možnost dostat se k dřívějšímu stavu daného svazku, přičemž data na původním svazku zůstanou ve stavu, v jakém jsou před zrušením snapshotu, tj. veškeré změny, které jste od pořízení snapshotu provedli, zůstanou součástí daného logického svazku. V této situaci by tedy obsah logického svazku vypadal následovně:

```
drwxr-xr-x   5 root root  4096 Feb 25 20:53 archive
drwxr-xr-x   7 root root  4096 Feb 25 19:15 cache
drwxr-xr-x 227 root root 12288 Feb 25 19:15 doc
```

Rušení snapshotu je velmi jednoduché - postačí použít nástroj lvremove a odebrat snapshot, jako by se jednalo o jakýkoliv jiný logický svazek:

```
lvremove vg/momentka
```

***Merge snapshotu***

Můžete se dostat do situace, kdy chcete provést rollback, tedy dostat obsah snapshotu zpět do logického svazku. Jak již bylo řečeno dříve, podmínkou je aktuální verze LVM a jádra, které zatím ještě nejsou v mnoha distribucích (např. v Debianu Lenny) k dispozici. Samotný merge zajišťuje nástroj lvconvertspuštěný s parametrem --merge a umístěním snapshotu:

```
[root@arch ~]# lvconvert --merge vg/momentka Merging of volume momentka started. snaptest: Merged: 9.6% snaptest: Merged: 0.0% Merge of snapshot into logical volume snaptest has finished. Logical volume "momentka" successfully removed
```

Samotný merge vyžaduje odpojený souborový systém na logickém svazku - pokud byste zadali výše uvedený příkaz a měli svazek nebo snapshot připojený, obdrželi byste následující chybovou hlášku:

```
[root@arch ~]# lvconvert --merge vg/momentka
Can't merge over open origin volume
Merging of snapshot momentka will start next activation.
```

LVM by pak jen vyčkal, až bude možné merge provést, a jakmile by byl oddíl odpojen, proces by zahájil. V příkladu, který jsem uváděl, by po provedení této operace vypadal souborový systém takto:

```
drwxr-xr-x   7 root root  4096 Feb 25 19:15 cache
drwxr-xr-x 227 root root 12288 Feb 25 19:15 doc
drwxr-xr-x   2 root root  4096 Feb 25 18:53 obsolete
```

***Snapshoty snapshotů***

Jelikož lze se snapshoty pracovat jako s obyčejnými blokovými zařízeními, včetně zápisu na ně, vyvstává v této souvislosti otázka, zda-li je možné vytvořit snapshot ze snapshotu. Bohužel, tuto funkcionalitu LVM v Linuxu zatím nemá. Je však možné vytvořit více snapshotů pro jeden logický svazek:

```
[root@arch ~]# lvcreate -s -L 2G -n momentka2 vg/snaptest
Logical volume "momentka2" created
[root@arch ~]# lvs
LV        VG   Attr   LSize Origin   Snap%  Move Log Copy%  Convert
momentka  vg   swi-a- 2.00g snaptest   0.00
momentka2 vg   swi-a- 2.00g snaptest   0.00
snaptest  vg   owi-ao 5.00g
```

***Co se stane, když se snapshot zaplní***

Do snapshotu se de facto zapisují všechny rozdíly vůči logickému svazku, ze kterého byl vytvořen, a to metodou "copy on write". V zásadě, kapacita snapshotu se snižuje jak vlivem změn ve snapshotu, tak vlivem změn v logickém svazku, kterému snapshot náleží. Pokud dojde k zaplnění celého snapshotu, začnou se dít psí kusy:

```
[root@arch ~]# lvs
/dev/dm-0: read failed after 0 of 4096 at 0: Input/output error
LV       VG   Attr   LSize  Origin   Snap%  Move Log Copy%  Convert
momentka vg   Swi-I-  2.00g snaptest 100.00
snaptest vg   owi-a-  5.00g
[root@arch ~]# mount -t ext3 /dev/vg/momentka /mnt
mount: wrong fs type, bad option, bad superblock on /dev/mapper/vg-momentka
...
```

Výpis hlášek jádra nástrojem dmesg začne ohlašovat nízkoúrovňové chyby na daném blokovém zařízení:

```
Buffer I/O error on device dm-0, logical block 1310718
EXT3-fs (dm-0): error: unable to read superblock
```

V tuto chvíli je již snapshot definitivně mrtvý, nedá se oživit, nedá se připojit, lze jej pouze odstranit:

```
lvremove vg/momentka
```

Pokud se ptáte, co se v takovém případě stane se souborovým systémem na původním logickém svazku, odpovědí je nic. Data na něm samozřejmě zůstanou. Horší situace nastane, pokud třeba používáte LVM pro kořenový oddíl, před upgradem systému vytvoříte jeho snapshot, abyste se mohli vrátit k původnímu systému, pokud se upgradem něco pokazí, následně provede upgrade, začnete s testováním a odhalíte fatální problém. Ještě než se pokusíte provést reintegraci (rollback) snapshotu pomocí nástroje lvconvert a jeho parametru --merge, pokusíte se získat co nejvíce informací o tom, co a proč se pokazilo. Mezi tím ale dojde místo na snapshotu a vy zůstanete s nefunkčním systémem.
Pokud budete používat snapshoty, pak je berte jako dočasné úložiště změn, o které můžete přijít, a původní logický svazek jako pevný bod, o který se můžete opřít. V situaci výše by bývalo stačilo, kdybyste před upgradem nastavili snapshot jako aktivní kořenový oddíl. Pak by přetečení snapshotu vyvolalo kolaps systému, ale vám by zůstal původní logický svazek ve stavu před upgradem.
Pokud má vaše distribuce novější LVM a jádro, pak máte možnost použít výše zmíněný merge, čímž se vám rozšiřují možnosti práce se snapshoty - už nemusíte uvažovat o snapshotu jako o snímku, který nakonec budete muset zahodit. V průběhu času se budete moci rozhodnout jak snímek zahodit a používat nadále logický svazek v jeho aktuálním stavu, tak provést "rollback" a změny provedené na logickém svazku zahodit.
Tím bych tento díl ukončil. Příště proberu používání zrcadlení na LVM a téma LVM definitivně uzavřu.

## Správa linuxového serveru: Ještě trocha LVM

### Zrcadlení v rámci LVM

Ačkoliv LVM vyniká flexibilitou práce s úložným prostorem, jeho velkým problémem je možné selhání fyzických svazků (physical volume), na kterých je LVM postaveno. Pokud tedy vytvořím skupinu svazků (volume group) z několika pevných disků, je sice na jednu stranu hezké, že k tomu prostoru mohu přistupovat jako ke spojitému úložnému prostoru, se kterým mohu zacházet jako s jedním velkým pevným diskem, který mohu dále libovolně dělit, ale co když některý z těch disků selže? LVM se v tomto ohledu chová podobně jako RAID 0, nemá žádnou redundanci a selhání fyzického svazku znamená ztrátu dat.

LVM v Linuxu však redundance přeci jen schopen je. V rámci LVM je totiž možné vytvořit logický svazek (logical volume), který bude fyzicky zrcadlen nad dvěma nebo více fyzických svazích. To je totéž, co provádí RAID 1. Předpokladem pro vytvoření zrcadleného svazku jsou alespoň tři (ano, opravdu tři) fyzické svazky. Je tomu tak kvůli nutnosti uložení diskového logu na třetí, nezávislé zařízení.

Na úvod vám ukážu tu nejjednodušší situaci, do které se můžete dostat:

```
[root@debian ~]# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/sdb   vg   lvm2 a-   10.00g 10.00g
  /dev/sdc   vg   lvm2 a-   10.00g 10.00g
  /dev/sdd   vg   lvm2 a-   10.00g 10.00g
```

Zde jsou vidět tři potřebné fyzické svazky s dostatečnou kapacitou. V takovém případě postačí vytvořit zrcadlený logický svazek, takto:

```
[root@debian ~]# lvcreate -L 9G -n zrcadlo -m 1 vg
  Logical volume "zrcadlo" created
```

Jakmile dojde k vytvoření zrcadleného oddílu, dojde i k zahájení úvodní synchronizace jednotlivých zrcadel. Je to proces velmi podobný jako v případě klasického RAIDu 1. Třetí disk, respektive malý kousek na něm, pak slouží jako synchronizační log, díky kterému bude možné udržet synchronizaci i po deaktivaci skupiny svazků (k té dojde třeba při restartu, apod.). Průběh synchronizace je možné sledovat pomocí nástroje lvs:

```
[root@debian ~]# lvs
  LV      VG   Attr   LSize Origin Snap%  Move Log         Copy%  Convert
  zrcadlo vg   mwi-a- 9.00g                zrcadlo_mlog   1.39
```

Pokud byste chtěli sledovat situaci v reálném čase, můžete si pomoci nástrojem watch:

```
watch -n 1 lvs
```

Tento příkaz bude de facto spouštět nástroj lvs každou sekundu. Můžete samozřejmě použít i časy kratší, i když v tomto případě to patrně nebude nutné.

#### *Co dělat, pokud máte pouze dva fyzické svazky*

První komplikací, která může nastat, je absence třetího fyzického svazku pro uložení mirror logů. V takovém případě, nevadí-li vám nutnost opětovné synchronizace při každém restartu počítače, můžete použít následující trik:

```
lvcreate -L 5G -n zrcadlo -m 1 vg --mirrorlog core
```

Můžete použít i kratší zápis --corelog místo --mirrorlog core. Obojí zajistí možnost vytvořit daný logický svazek pouze na dvou fyzických svazcích místo třech.

#### *Nedostatek souvislého prostoru na fyzických svazích*

Druhou možnou komplikací může být nedostatek souvislého prostoru na fyzických svazcích. Pro lepší ilustraci, uvažte následující situaci:

```
[root@debian ~]# lvs
  LV      VG   Attr   LSize Origin Snap%  Move Log Copy%  Convert
  root    vg   -wi-a- 8.00g
  storage vg   -wi-a- 8.00g
[root@debian ~]# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/sdb   vg   lvm2 a-   10.00g  2.00g
  /dev/sdc   vg   lvm2 a-   10.00g 10.00g
  /dev/sdd   vg   lvm2 a-   10.00g  2.00g
```

Pokud se podíváte na dostupnou kapacitu skupiny svazků, zjistíte, že je možné vytvořit až 7GB velký zrcadlený logický svazek, ale volné místo na discích sdb a sdd je po dvou GB. Pokud předpokládáte, že v této konfiguraci postačí zadat LVM vytvoření zrcadleného oddílu a on potřebné místo uvolní, pak vás musím zklamat, tohle LVM automaticky provést neumí:

```
[root@debian ~]# lvcreate -L 6G -n zrcadlo -m 1 vg
  Insufficient suitable allocatable extents for logical volume : 1025 more required
  Unable to allocate extents for mirror(s).
```

Dejme tomu, že vám bude stačit 6GB. Abyste tedy mohli zrcadlený svazek vytvořit, musíte uvolnit potřebné místo na jednotlivých fyzických svazcích ručně. K tomu použijete nástroj pvmove. Chcete-li 6GB velký svazek a máte-li v dané skupině svazků velikost fyzického extentu (PE) 4MB, budete potřebovat alespoň 1536 PE na dvou fyzických svazcích, neboť:

```
(6 * 1024) / 4 = 1536
```

Zjistit konkrétně hodnoty alokace PE můžete pomocí nástroje vgdisplay:

```
[root@debian ~]# pvdisplay /dev/sdb
  --- Physical volume ---
  PV Name               /dev/sdb
  PE Size               4.00 MiB
  Total PE              2559
  Free PE               511
  Allocated PE          2048
```

Jelikož v tomto případě potřebujete 1536 volných PE, přesunete PE od extentu 1022 až po poslední alokovaný extent, tedy 2048, ze svazku sdb na svazek sdc:

```
pvmove /dev/sdb:1022-2048 /dev/sdc
```

Totéž provedu se svazkem sdd:

```
pvmove /dev/sdd:1022-2048 /dev/sdc
```

Výsledek bude vypadat takto:

```
[root@debian ~]# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/sdb   vg   lvm2 a-   10.00g 6.00g
  /dev/sdc   vg   lvm2 a-   10.00g 1.99g
  /dev/sdd   vg   lvm2 a-   10.00g 6.00g
```

Nyní již můžete konečně vytvořit požadovaný zrcadlený logický svazek:

```
[root@debian ~]# lvcreate -L 6G -n zrcadlo -m 1 vg
  Logical volume "zrcadlo" created
```

Ruční žonglování s PE a nutnost řešit související matematické úlohy asi není to pravé ořechové, ale bohužel mi kromě možnosti napsat si za tímto účelem nějaký pomocný shellový skript není znám žádný rychlejší postup.

#### *Když disk selže...*

Pokud disk selže, projeví se to ve výpisu fyzických svazků takto:

```
[root@debian ~]# pvs
  Couldn't find device with uuid 'fM74J2-9Utu-jvCL-mGVg-7xJW-iiQd-l0Lvgf'.
  PV           VG   Fmt  Attr PSize  PFree
  /dev/sdc     vg   lvm2 a-   10.00g 1.98g
  /dev/sdd     vg   lvm2 a-   10.00g    0
```

```
unknown device vg   lvm2 a-   10.00g    0
```

Pokud samotný zápis na zrcadlený svazek nevyvolá automatickou konverzi zrcadleného svazku na lineární (nezrcadlený), lze to učinit ručně, odebráním chybějících zařízení ze skupiny svazků:

```
[root@debian ~]# vgreduce --removemissing vg --force
Couldn't find device with uuid 'fM74J2-9Utu-jvCL-mGVg-7xJW-iiQd-l0Lvgf'.
WARNING: Bad device removed from mirror volume, vg/zrcadlo
WARNING: Mirror volume, vg/zrcadlo converted to linear due to device failure.
Wrote out consistent volume group vg
WARNING: dev_open(/etc/lvm/cache/.cache) called while suspended
```

Po výměně disku postačí nový disk vložit do skupiny svazků:

```
[root@debian ~]# pvcreate /dev/sdb
Physical volume "/dev/sdb" successfully created
[root@debian ~]# vgextend vg /dev/sdb
Volume group "vg" successfully extended
```

A následně provést konverzi nyní lineárního svazku "zrcadlo" na svazek zrcadlený:

```
lvconvert -m1 vg/zrcadlo
```

**Přístup k LVM ze záchranného/cizího systému**

LVM si uchovává podstatné údaje jednak v /etc/lvm a jednak v hlavičkách fyzických svazků. Pokud máte celý systém na LVM a dojde k nějaké havárii a systém vám nenastartuje, pak musíte LVM připojit ručně.

Nástroje `pvscan`, `vgscan` a `lvscan` po svém spuštění bez parametru osahají všechny dostupné disky a najdou všechny fyzické svazky (physical volumes), skupiny svazků (volume groups) a logické svazky (logical volumes).

Abyste vytvořili všechna potřebná zařízení v /dev, použijte následující příkaz:

```
vgchange -ay
```

Pak budete moci přistupovat ke všem logickým svazkům, připojit je a provést případnou záchranou operaci.

**Strategie použití LVM na serveru**

Pokud budete spravovat server (a nejenom v tomto případě), pak se rozhodně vyplatí alokovat jednotlivým logickým svazkům co nejméně, nechat si ve skupině svazků volné místo a spíše logické svazky rozšiřovat dle potřeby, než alokovat celý dostupný prostor a pak přemýšlet, co zmenšit. Volný prostor je nutný nejen k vytváření dalších logických svazků, ale i k vytváření snapshotů.

***Zrcadlit pomocí LVM nebo RAIDem 1?***

Podle mého názoru je nejideálnější schéma nasazení LVM nad nějakým RAIDem, ať už RAIDem 1, 5, 6 nebo třeba 10. Samotné zrcadlení v rámci LVM je zajímavé, ale trpí jistými drobnými nedostatky, počínaje nutností mít tři fyzické svazky. Také je nutné počítat s tím, že LVM je co do funkčnosti podstatně rozsáhlejší a složitější než samotný RAID 1, ať už softwarový či hardwarový. V případě problémů (hardwarových i softwarových) je RAID 1 přeci jen jednodušší v případě záchranných operací.

Když už zmiňuji RAID a zrcadlení, neodpustím si zopakovat svou obligátní poznámku, že tyto technologie nejsou náhražkou zálohování, ba právě naopak. S každou funkcionalitou navíc rostou možnosti nějakého selhání či problému. V souvislosti s LVM je určitě dobré zmínit jeden velice zajímavý článek o jeho vztahu k souborovým systémům.

Tím bych problematiku LVM uzavřel. V příštím dílu seriálu se podívám na dm-crypt/LUKS z praktického hlediska.

## Správa linuxového serveru: Šifrování s dm-crypt/LUKS

### Co lze šifrovat a proč

Pomocí dm-crypt/LUKS lze šifrovat **jakékoliv blokové zařízení**, tzn. diskový oddíl, celý pevný disk, diskové pole (RAID), logický svazek v rámci LVM, atd.

Připomínám, že i když se to na první pohled možná nezdá, šifrování jako takové **nelze** považovat za spolehlivý nástroj pro zajištění bezpečnosti dat před případným útočníkem s fyzickým přístupem k serveru. Nemůže tedy nahradit strážného, který hlídá přístup do serverovny. Důvody jsem popsal již dříve. Může nicméně pomoci ochránit data při likvidaci nebo reklamaci pevných disků, může pomoci zabránit úniku SSH klíčů nebo jiných citlivých údajů ze zcizených laptopů administrátorů, apod.

Jelikož se v současné době začínají masivně rozrůstat možnosti virtualizace, je nutné uvažovat o bezpečnosti takových řešení, a naprosté zbytečnosti šifrování jako ochrany před poskytovatelem virtuálního serveru, jelikož šifrovací klíče jsou uloženy v paměti, a do paměti má obvykle správce virtuálního serveru přístup. Budete-li tedy uvažovat o nasazení šifrování, uvažte pečlivě, jaká data chcete chránit a před kým. Následně zauvažujte, zda-li má šifrování v daném scénáři smysl, a pokud ano, tak jak velký. Jelikož má šifrování negativní dopad na výkon i na případnou záchranu dat z poškozeného pole nebo disku, je třeba pečlivě zvážit, zda-li šifrování nasadíte a kam.

### Co šifrovat

Aby mělo použití šifrování smysl, je třeba zajistit, aby na discích nezůstalo nic z chráněných dat v nešifrované podobě. Proto je každopádně třeba začít se šifrováním swapu. Dále je třeba kromě samotných úložišť citlivých dat zabezpečit i všechna místa, kam by se data mohla dostat. Typicky je to adresář /tmp, pro který bývá dobré využívat buď tmpfs nebo samostatný šifrovaný oddíl. V některých specifických případech to může být kromě obvyklého /home i adresář /var, /srv či jiné umístění. Šifrovat lze samozřejmě jak selektivně (tj. jen to důležité), tak téměř úplně všechno včetně celého systému s výjimkou zavaděče, obrazu jádra a initrd. V prvním případě je nutné dát si pozor na "prosakování" chráněných dat mimo šifrované svazky, ve druhém případě obvykle vznikne problém se zaváděním systému, během kterého bude třeba nějakým způsobem zadat heslo, což třeba u serveru, který je umístěn v nějakém datacentru, a ke kterému přistupujete vzdáleně, působí jistý problém. Ten je možné řešit pomocí malého ssh démona v initrd (třeba dropbear), který se spustí a umožní správci přihlásit se a zadat heslo vzdáleně.

Návod pro toto řešení v rámci distribuce Debian naleznete v článku na debian-administration.org.

### Výběr šifry a šifrovacího módu

Před samotným nasazením šifrování je třeba připomenout problematiku šifrovacích algoritmů a šifrovacích módů. V zásadě, šifrovací algoritmus byste měli vybírat nejlépe z finalistů AES. Starším šifrám jako DES či Blowfish se doporučuji vyhnout. Se samotným výběrem vám neporadím, jen podotknu, že nejrychlejší implementací je samotný AES/Rijndael, v závěsu za ním pak Serpent a Twofish. Jejich bezpečnost by měla být velmi podobná. Pro diskové šifrování je klíčový výběr vhodného šifrovacího módu. Je totiž nutné šifrovat ohromné kvantum dat, která mají ovšem předvídatelnou strukturu (souborový systém, superblok, atd.), což může případnému útočníkovi usnadnit kryptoanalýzu, i když je samotný šifrovací algoritmus bezpečný. Pro diskové šifrování dnes přichází v úvahu především mód XTS. LRW je vzhledem k jeho objevené zranitelnosti nutné zavrhnout. Pro systémy se staršími jádry, které ještě nemají podporu pro mód XTS, je vhodné použít mód cbc-essiv (nikoliv však samotné cbc!). Řada distribucí nabízí v rámci instalátoru možnost postavit šifrovaný systém. Bohužel, tato možnost obvykle neumožňuje precizní nastavení šifrovacího algoritmu a šifrovacího módu, a mnohde bývají výchozí hodnoty nastavené patrně ve snaze zachovat kompatibilitu příliš konzervativně (což je i případ Debianu), tudíž není zvolen výchozí mód XTS, ale třeba cbc-essiv. V takových případech je tedy vhodnější postavit šifrovaný systém ručně, neboť šifrovací mód na existujícím šifrovaném svazku měnit nelze.

### Alternativy pro šifrování v GNU/Linuxu

Pro diskové šifrování v GNU/Linuxu je ideální používat dm-crypt/LUKS, nicméně v oblasti šifrování to není jediný prostředek, který mají uživatelé k dispozici. Představím vám několik alternativ, ať už pro diskové šifrování, tak pro šifrování obecně.

#### *dm-crypt*

Asi nejjednodušší alternativou dm-crypt/LUKS je dm-crypt jako takový. Jeho asi jedinou výhodou (chcete-li se na to tak dívat) oproti jeho kombinaci s LUKSem je absence hlavičky, která dané blokové zařízení nezaměnitelně označí jako šifrované. Oproti LUKS nadstavbě má však řadu citelných nevýhod, počínaje nutností celý oddíl přešifrovat při změně hesla přes nemožnost použít více než jedno heslo až po absenci posílení hesla metodou PBKDF2.

#### *Truecrypt*

Nejvíce zmiňovanou alternativou pro dm-crypt/LUKS je Truecrypt, zejména kvůli přenositelnosti šifrovaných kontejnerů mezi GNU/Linuxem a MS Windows. V dnešní době je Truecrypt schopen využívat přímo linuxové CryptoAPI a vytváří přímo blokové zařízení prostřednictvím Device Mapperu, což jej řadí blízko k dm-cryptu a LUKS. Samotný Truecrypt nabízí uživatelům pěkné grafické rozhraní, stejně jako možnost jej ovládat přes příkazový řádek. Jeho výhodou oproti dm-crypt/LUKS jsou skryté svazky uvnitř šifrovaného svazku a s tím související "plausible deniability" aneb možnost "hodnověrného popření" - skryté svazky využívají steganografii, měly by tudíž být neviditelné, pokud připojíte pouze samotný šifrovaný svazek (a nikoliv skrytý svazek v něm ukrytý). V Truecryptu pro GNU/Linux je tato funkcionalita podporována, ale pouze s použitím souborového systému FAT, což může být o něco "nápadnější" než v případě MS Windows. Obzvláště pak, když Truecrypt jinak bez problémů umožňuje vytvářet kontejnery s Ext2/Ext3. Je nutné dodat, že licence Truecryptu je mnohými distribucemi (Debian, Fedora a další) považována za nesvobodnou a mj. pro distributory potenciálně problematickou. Proto třeba nenajdete Truecrypt v oficiálních repositářích Debianu.

#### *encfs*

Další alternativou je encfs (web projektu) využívající FUSE. Jeho výhodou i nevýhodou zároveň (záleží na úhlu pohledu) je funkce "nad" existujícím souborovým systémem, nikoliv "pod" ním jako v případě ostatních zmíněných řešení. Šifrují se tedy přímo vlastní data, včetně názvů souborů a adresářů. Potenciálním problémem je zachování metadat (velikost souboru, datum vytvoření a poslední modifikace, atd.) pro případného útočníka. To sice nemá přímý bezpečnostní dopad ve smyslu usnadnění kryptoanalýzy, ale může to v některém případě odhalit jisté informace, které by nebyly k dispozici, kdyby byl zašifrovaný celý souborový systém.

Encfs se však výborně hodí pro různé síťové disky a podobné online služby, kde příslušná data nemáte pod kontrolou a nevíte, kdo má k nim přístup. Jistě, je možné použít šifrovaný kontejner, ale to nemusí být zcela efektivní z hlediska využitého prostoru, který je v rámci daných služeb obvykle velmi omezený.

#### *GnuPG*

GnuPG je běžnou součástí distribucí a umožňuje jak šifrování souborů asymetrickým klíčem, tak šifrování běžnou, blokovou šifrou. Pomocí GnuPG lze tedy snadno zašifrovat nějaký soubor buď k přenosu přes nezabezpečenou síť nebo pro uložení na nějaké médium či síťový disk.

GnuPG využívá, kupříkladu, zálohovací nástroj Duplicity, který umožňuje vytvářet na místním či vzdáleném úložišti šifrované zálohy.

### Jak poskládat RAID, LVM a dm-crypt/LUKS?

Na závěr tohoto dílu navážu na předchozí tři díly, které se věnovaly LVM. Pokud se podíváme na možnosti uložení, zabezpečení a organizace dat, máme bloková zařízení (pevné disky), z těch můžeme vytvářet disková pole, z disků i diskových polí pak můžeme vytvářet skupiny svazků a logické svazky, přičemž šifrovat můžeme v podstatě kdekoliv.

Základ v každém případě tvoří pevné disky nebo disková pole. U serverů je velmi vhodné použít diskové pole s redundancí k zajištění jisté odolnosti vůči výpadku. V případě většího množství disků, kde je velmi vhodné použít různé disky od různých výrobců, je možné použít RAID 10 (RAID 0 na jednotlivých párech zrcadlených disků). RAID 10 nebo 01 dosahuje optimálního kompromisu mezi výkonem a redundancí, ale potřebuje minimálně 4 disky.

LVM je velmi vhodné provozovat z hlediska bezpečnosti dat nad diskovým polem s redundancí a používat jej spíše jen k rozdělení dostupného prostoru do logických svazků, snapshotům, apod. dm-crypt/LUKS je ideální umístit buď mezi diskové pole a LVM nebo nad LVM (šifrovat logický svazek). Přidáním dalšího disku nebo disků lze pak postupně krok za krokem dostupný prostor rozšířit (dm-crypt/LUKS podporuje operaci resize a u LVM to zvládne pvresize). Šifrovat lze samozřejmě i pod softwarovým RAIDem, což bývá i doporučeno, jsou-li pevné disky příliš velké (příliš mnoho dat šifrovaných jedním postupem usnadňuje případnou kryptoanalýzu). Nutnost "odemknout" více zařízení může vyřešit malý šifrovaný oddíl na jednom z disků, který obsahuje klíče k "odemčení" zbylých disků v poli. Co se týče ostatních kombinací, provozovat softwarový RAID nad LVM je sice možné, ale naprosto nesmyslné a potenciálně problémové, a provozovat LVM bez RAIDu na více discích je nevhodné - stoupá pravděpodobnost selhání jednoho z disků a selhání kteréhokoliv z disků znamená ztrátu dat. Pokud byste používali zrcadlení v rámci LVM, je podle mého lepší využít rovnou softwarový nebo hardwarový RAID.

Tím bych tento díl ukončil. Příště už proberu dm-crypt/LUKS z ryze praktického hlediska.

## Správa linuxového serveru: Praxe šifrování s dm-crypt/LUKS

Minule jsem vás uvedl to teorie diskového šifrování, dnes se na toto téma podívám z praktického hlediska.

### dm-crypt/LUKS

dm-crypt je specifický subsystém linuxového jádra, který využívá infrastrukturu device mapperu a linuxového CryptoAPI, a vytváří transparentní vrstvu pro online šifrování a dešifrování blokových zařízení. Dodávám, že dm-crypt je možné využívat samostatně, bez LUKS nadstavby, a to prostřednictvím stejného nástroje - cryptsetup. V tomto článku se však plně zaměřím právě na LUKS.

LUKS je nadstavba dm-cryptu, která, vezmu-li to zjednodušeně, přidává dvouúrovňové šifrování, správu klíčů a funkci pro posílení uživatelského hesla **PBKDF2**. Specifikace LUKS je multiplatformní a je možné s jeho pomocí vytvářet šifrované kontejnery, které lze otevřít i v jiných operačních systémech (třeba s pomocí aplikace pro MS Windows s názvem FreeOTFE).

Dvouúrovňové šifrování v případě LUKS znamená, že samotný svazek je šifrován tzv. hlavním klíčem (master key), který je následně zašifrován uživatelským heslem a uložen do jednoho z úložišť (slotů). Z toho vyplývá jak na jedné straně velká výhoda v podobě nezávislosti šifrování na uživatelském hesle, a tudíž možnost bezproblémové revokace (či změny) existujícího přístupového hesla, tak na straně druhé nevýhoda spočívající v potenciální ztrátě dat, je-li nezálohovaná hlavička s úložištěm klíčů přepsána.

### Příprava budoucího šifrovaného svazku

Ještě než začnete s vytvářením šifrovaného svazku, je dobré příslušné blokové zařízení zaplnit co nejkvalitnějšími náhodnými daty. Je to z toho důvodu, aby případný kryptoanalytik viděl na daném blokovém zařízení jednu celistvou oblast a neměl možnost uhodnout, které bloky jsou opravdu šifrované a které ne (zašifrovaná data by měla být neodlišitelná od náhodných dat), což by mu mělo o něco více zkomplikovat jeho práci. Tento krok sice není nutný, ale bývá vhodné ho provést.

Zcela ideální k zaplnění svazku náhodnými daty by bylo použít generátor náhodných čísel /dev/random. Tento generátor vytváří kvalitní náhodná čísla, ale je neuvěřitelně pomalý. Z tohoto důvodu je podstatně vhodnější použít zařízení /dev/urandom, které sice generuje méně "kvalitní" náhodná čísla, ale generuje je neporovnatelně rychleji.

K samotnému zaplnění zvoleného svazku náhodnými daty použijte nástroj dd:

```
dd if=/dev/urandom of=/dev/svazek
```

Tato procedura bude obvykle trvat velmi dlouho, řádově hodiny až dny. Existují sice metody, jak tuto fázi uspíšit, ale jejich důsledkem je zaplnění svazku méně kvalitními náhodnými daty, které patrně může zkušený kryptoanalytik rozeznat od šifrovaných dat, což by činilo celý tento krok zbytečným.

Já osobně používám následující urychlovací metodu. Dodávám však, že nejsem kryptolog ani kryptoanalytik, tudíž nejsem schopen odhadnout, zda-li je to správný postup nebo naopak z nějakého důvodu velmi špatný.

Můj postup je následující - vytvořím šifrovaný svazek (viz další bod), následně použiji nástroj badblocks k zaplnění svazku nekvalitními náhodnými daty, která by ovšem po zašifrování měla být nerozeznatelná od kvalitních (nebo od skutečných zašifrovaných dat):

```
badblocks -v -s -w -t random /dev/mapper/desifrovany_svazek
```

Následně svazek odpojím pomocí luksClose (viz níže) a přepíšu ještě začátek oddílu s LUKS hlavičkou pomocí /dev/urandom:

```
dd if=/dev/urandom of=/dev/zarizeni bs=1M count=32
```

### Vytvoření šifrovaného svazku

Ke všem operacím se šifrovanými svazky budeme využívat nástroje cryptsetup. Ten byste měli mít k dispozici v repositářích vaší distribuce (v případě Debianu se tento nástroj nachází ve stejnojmeném balíčku). Vytvoření šifrovaného svazku provedete s použitím šifrovacího algoritmu AES a módu XTS takto:

```
cryptsetup -c aes-xts-benbi -s 512 -y luksFormat /dev/svazek
```

Povšimněte si parametru -c, kde je na prvním místě šifrovací algoritmus, na druhém, oddělen pomlčkou, šifrovací mód a na třetím způsob generování inicializačního vektoru (v případě XTS máte na výběr mezi 32-bitovým "plain" a 64-bitovým "benbi"). Parametr -s udává délku klíče, -y vám umožní zadat heslo pro kontrolu dvakrát a luksFormat označuje LUKS operaci, tedy inicializaci daného svazku.

V případě, že máte k dispozici starší systém, který nezná XTS, použijte mód cbc-essiv:

```
cryptsetup -c aes-cbc-essiv:sha256 -s 256 -y luksFormat /dev/svazek
```

### Otevření zašifrovaného svazku

Po tom, co zašifrujete nějaký svazek, budete jistě chtít svazek otevřít a přistupovat k němu jako k nešifrovanému, abyste na něm mohli vytvořit souborový systém, ten připojit a nahrát na něj nějaká data. Za tímto účelem použijete LUKS operaci open:

```
cryptsetup luksOpen /dev/svazek odsifrovano
```

Prvním parametrem je samotné zařízení, druhým pak název, pod kterým bude k dispozici dešifrované zařízení umístěné v /dev/mapper. Po této operaci byste tedy měli mít (za předpokladu zadání správného hesla) k dispozici dešifrované zařízení /dev/mapper/odsifrovano, na kterém pak pomocí mkfs vytvoříte souborový systém a zařízení připojíte.

### Zrušení mapování

Jakmile skončíte práci s namapovaným dešifrovaným svazkem, bývá dobré příslušné mapování zrušit. K tomu slouží LUKS operace close:

```
cryptsetup luksClose odsifrovano
```

Předpokladem je, že před tím odpojíte příslušný souborový systém. Povšimněte si také, že luksCloseoperuje s názvy dešifrovaných svazků, nikoliv s původními zařízeními.

### Správa klíčů

Jelikož LUKS oplývá vlastností dvouúrovňového šifrování, máte možnost nastavit více klíčů, přičemž každý pak bude schopen daný svazek dešifrovat. Ke správě klíčů slouží operace luksAddKey (přidá nový klíč) aluksRemoveKey (odstraní stávající klíč).

Po vytvoření šifrovaného svazku pomocí luksFormat již svazek bude mít definováno jeho heslo. Pokud budete chtít později přidat další klíč, zadáte:

```
cryptsetup luksAddKey /dev/svazek
```

LUKS vás vyzve k zadání jakéhokoliv jiného hesla (aby mohl získat hlavní klíč), a pak vás nechá zadat heslo nové. Tím pak hlavní klíč zašifruje a uloží do některého z volných "slotů" v úložišti pro klíče.

Všimněte si, že se zde používá název původního zařízení, není tedy nutné před přidáním či odebráním klíče svazek dešifrovat pomocí luksOpen.

Analogicky, budete-li chtít některý z existujících klíčů odebrat, použijete:

```
cryptsetup luksRemoveKey /dev/svazek
```

Operace pro odebrání klíče je naštěstí dostatečně inteligentní, aby uživateli zabránila odebrat poslední klíč nebo odebrat klíč bez znalosti nějakého dalšího klíče.

Kromě hesel můžete samozřejmě používat i klíčové soubory (keyfile). V případě operací luksAddKey aluksRemoveKey postačí uvést klíčový soubor za názvem operace, takto:

```
cryptsetup luksAddKey /cesta/ke/keyfile
```

A analogicky pro odebrání klíče:

```
cryptsetup luksRemoveKey /cesta/ke/keyfile
```

Jelikož klíčový soubor postačí sám o sobě k dešifrování daného svazku, je třeba věnovat zvýšenou pozornost jeho bezpečnému uložení. V ideálním případě by se měl nacházet na jiném šifrovaném svazku.

Tím bych tento díl ukončil. Příště proberu zálohu LUKS hlavičky, začlenění šifrovaných svazků do systému a šifrování celého systému.

## Správa linuxového serveru: Dokončení praxe šifrování s dm-crypt/LUKS

Minule jsem začal s praxí šifrování s dm-crypt/LUKS, dnes toto téma dokončím. Budu se věnovat například šifrování celého systému.

### Záloha hlavičky

Pro uložení základních informací o šifrovaném svazku využívá LUKS hlavičku, kterou vytváří na začátku šifrovaného svazku. Zde se nachází mj. i úložiště klíčů. Jelikož hlavní klíč (master key) sloužící k dešifrování LUKS svazku leží právě v úložišti klíčů (přesněji, je zašifrován jedním nebo více uživatelskými klíči), toto umístění je nejslabším článkem řetězu. Případné přepsání hlavičky znamená jednoznačně ztrátu všech dat na šifrovaném svazku, nemáte-li někde po ruce zálohu.

Zálohovat hlavičku možné samozřejmě je, avšak je třeba dodat, že s příslušnou zálohou je třeba zacházet opatrně - pokud se časem rozhodnete revokovat některý z přístupových klíčů, třeba z důvodu jeho kompromitace, máte stále zálohu, kde je hlavní klíč šifrovaný oním revokovaným klíčem. Proto je třeba zvážit, zda-li se vám záloha hlavičky vůbec vyplatí vzhledem k bezpečnosti, a pokud ano, je jednoznačně třeba takovou zálohu bezpečně uložit.

Abyste mohli zálohu provést, musíte zjistit, jak velká hlavička je. Za tímto účelem stačí údaje z hlavičky vypsat:

```
# cryptsetup luksDump /dev/svazek

LUKS header information for /dev/svazek

Version:        1
Cipher name:    aes
Cipher mode:    cbc-essiv:sha256
Hash spec:      sha1
Payload offset: 2056
MK bits:        256
MK digest:      63 7b 35 48 ec 3d e6 f5 fb 83 04 22 75 22 d5 4a aa 17 bb 75
MK salt:        2a 01 a3 fa d2 96 17 c9 5a 4c 2e 63 72 27 1c 70
                25 1f e6 27 2e 7a 4c 7c f7 55 fe 8e af ee 1f 2d
MK iterations:  10
UUID:           0007771a-83e7-4256-b31b-d6a5cbb41663

Key Slot 0: ENABLED
        Iterations:             211800
        Salt:                   44 82 02 d6 f5 87 41 d9 1c 76 9c aa a3 88 85 8c
                                71 a9 31 22 a3 15 3f b4 c6 e5 c1 76 4b 30 c2 c6
        Key material offset:    8
        AF stripes:             4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

V tomto výpisu si povšimněte položky Payload offset, která činí v mém případě 2056 bloků (ve vašem případě může být hodnota jiná). Tuto hodnotu následně dosaďte do následujícího příkazu:

```
dd if=/dev/svazek of=zaloha.hlavicky count=2056
```

Pro obnovu hlavičky postačí prohodit if a of.

Pokud se vrátím zpět k výpisu LUKS hlavičky, upozorním vás ještě na několik věcí. Z výpisu je jasně patrná šifra (cipher name), šifrovací mód (cipher mode), použitá hash a obsazení jednotlivých "zásuvek" (slotů) v úložišti pro klíče.

Budete-li váhat, proč i po tom, co si zvolíte za hash při operaci luksFormat něco jiného než SHA1, zůstane v kolonce "hash spec" stále SHA1 místo vámi zvolené hashe, je to kvůli implementaci PBKDF2, funkci pro posílení uživatelského hesla. Ta pracuje tak, že z uživatelského hesla získá hash, poté získá hash z takto vzniklé hashe, a toto se mnohokrát opakuje v cyklu (v případě klíče ve slotu 0 to bylo 211800 iterací - viz výpis výše). Tím se trošku zkomplikuje brute froce útok na dané heslo, protože každý pokus pak musí projít stejným počtem iterací.

Naneštěstí je v implementaci funkce PBKDF2 u LUKS na pevno nastavena hash SHA1, takže ať už si zvolíte jakoukoliv hash, v rámci LUKS se pro klíč bude používat SHA1.

### Dešifrování zařízení při bootu

U některých svazků se vám určitě hodí jejich inicializace při startu (při šifrování celého systému se tomu nevyhnete). K tomu slouží v řadě distribucí konfigurační soubor /etc/crypptab. Bohužel, jeho syntax se mezi distribucemi může lišit. V Debianu je syntax následující:

```
# <target name>         <source device>                  <key file>    <options>
cswap /dev/sda1 /dev/urandom swap
chome /dev/sda3 none luks
```

Prvním parametrem je název dešifrovaného svazku, který bude k dispozici v /dev/mapper po úspěšném "odemčení" svazku. Druhým parametrem je zdrojové zařízení, na kterém je šifrovaný svazek. Třetí parametr označuje klíčový soubor. Všimněte si, že v případě šifrovaného swapu je možné generovat klíč náhodně při každém startu pomocí generátoru náhodných čísel /dev/urandom. Posledním parametrem jsou volby, kterými je možné inicializaci šifrovaného svazku řídit. Tou nejdůležitější pro vás je asi volba luks, která označuje LUKS svazek. Bez udání této volby se použije rozhraní dm-crypt bez LUKS rozšíření.

Inicializace šifrovaných svazků probíhá při zavádění systému, tudíž vás systém může vyzvat k zadání hesla už v této fázi. Samotné připojení souborového systému na dešifrovaném svazku je třeba obstarat ve fstab:

```
/dev/mapper/cswap swap  swap defaults         0 0
/dev/mapper/chome /home ext3 defaults,noatime 0 0
```

### Šifrování celého systému

Šifrování celého systému možné je, s výjimkou obrazu jádra, initrd a zavaděče. To jsou komponenty, které musí zůstat nešifrované, jinak se systém nezavede. Z tohoto důvodu je třeba izolovat samostatný oddíl pro/boot. Vše ostatní včetně kořenového adresáře je možné šifrovat. Zejména na laptopech bývá kvůli hibernaci ideální schovat swap (kam se ukládá obraz paměti) do LVM nad zašifrovaným svazkem.

V případě šifrování celého systému se nevyhnete vytvoření iniciálního ramdisku (initrd), kde musí být mj. samotný nástroj cryptsetup (moduly pro souborový systém, diskové řadiče a šifrování mohou být samozřejmě zakompilovány do jádra). V rámci initrd je třeba zajistit načtení nutných modulů, což jsou moduly zahrnující modul se šifrovacím algoritmem (např. aes-i586), modul s šifrovacím módem (např. xts), dále pak dm-crypt a dm-mod.

V Debianu se naštěstí tato konfigurace provádí z velké části automaticky. Už po instalaci cryptsetupu dojde k automatické úpravě iniciálního ramdisku, který získá podporu pro dešifrování kořenového svazku. Tento "automat" však nefunguje úplně vždycky korektně, resp. ne vždy pochopí, co správce zamýšlí.

Předně, v době generování iniciálního ramdisku je třeba mít zavedené všechny moduly potřebné pro dešifrování kořenového svazku (ideálně by daný svazek měl být připojen). Dále je třeba adekvátně upravit/etc/crypttab a /etc/fstab. Poté je třeba nechat znovu vygenerovat iniciální ramdisk a upravit zavaděč, aby jádru předal vhodný parametr root= (třeba /dev/mapper/root).

Instalátor Debianu umí vytvořit šifrovaný systém i s kořenovým svazkem. Problémem je, jak už bylo zmíněno dříve, nastavení, které neumožňuje použít lepší šifrovací mód než cbc-essiv. Jelikož šifrovací mód nelze na existujícím šifrovaném svazku jen tak změnit, je lepší celý proces provést ručně.

Já se zaměřím na situaci, kdy již máte systém nainstalovaný, a chcete jej zašifrovat. Prvním krokem by mělo být šifrování swapu - tento krok je již naznačen výše.

Samotná příprava a vytvoření šifrovaného svazku byla probrána v minulém díle. Varoval bych vás snad jen před různými návody na rychlé zašifrování nešifrovaného oddílu pomocí nástroje dd. Uvědomte si, že operace luksFormat přepíše začátek daného oddílu a samotná LUKS hlavička oddíl o něco zmenší (tj. souborovému systému by to uřízlo konec).

Druhým krokem může být zašifrování všech ostatních svazků s výjimkou kořenového. Bohužel mi není známa spolehlivá metoda zašifrování nešifrovaného svazku (alespoň ne, jedná-li se o LUKS svazek). Z tohoto důvodu se asi nevyhnete přesunům dat mimo budoucí šifrovaný svazek a pak zpět, což ovšem není z hlediska bezpečnosti vůbec ideální, jelikož citlivá data se během tohoto procesu přesouvají na další médium - toto médium, stejně jako i původní svazek, je třeba ošetřit tak, aby citlivá data již nebylo možné obnovit (viz bezpečné mazání v závěru článku).

Třetím krokem je příprava zašifrovaného kořenového svazku. Ideální je, pokud máte možnost budoucí šifrovaný svazek připojit přímo za běhu (moduly do iniciálního ramdisku se pak načtou automaticky). Pokud tuto možnost nemáte, je třeba přidat všechny potřebné moduly (viz výše) do konfiguračního souboru/etc/initramfs-tools/modules. Předpokladem je nainstalovaný balíček initramfs-tools.

Dále je třeba vytvořit záznam v konfiguračním souboru /etc/crypttab:

```
root /dev/zarizeni none luks
```

Upravte si /dev/zarizeni na disk či oddíl pro zašifrovaný kořenový svazek.

Adekvátně je třeba upravit /etc/fstab. Tam můžete mít, kupříkladu, následující řádek:

```
/dev/hda3       /    ext3    noatime,errors=remount-ro 0       1
```

Původní zařízení (v tomto výpisu by to bylo /dev/hda3) nahraďte cestou k blokovému zařízení "dešifrovaného" svazku, tedy /dev/mapper/root. Výsledek by měl vypadat takto:

```
/dev/mapper/root /    ext3    noatime,errors=remount-ro 0       1
```

V tuto chvíli by mělo být téměř vše připraveno, tudíž můžete zkusit vygenerovat iniciální ramdisk, a to provedete následujícím příkazem:

```
update-initramfs -k all -u
```

Následuje poslední úprava - úprava konfigurace zavaděče, kde je třeba kořenový oddíl (typicky je předávaný jádru v hodnotě parametru root=) zaměnit za /dev/mapper/root.

Nyní již zbývá data z kořenového oddílu přesunout, z oddílu vytvořit šifrovaný svazek a na ten data přesunout zpět. Nejpohodlněji se to provádí z nějaké live distribuce, třeba z mého oblíbeného System Rescue CD.

Poté proveďte reboot, a pokud jste nastavili všechno správně, systém by měl naběhnout, dotázat se na příslušná hesla a zpřístupnit šifrované svazky.

## Bezpečné mazání

V souvislosti se šifrováním by bylo dobré probrat také bezpečné mazání, respektive problém související s tím, jak se bezpečně zbavit citlivých dat tak, aby nebylo možné je obnovit, zejména pak z magnetických záznamových zařízení (jako jsou např. pevné disky), ze kterých je teoreticky možné se speciálním vybavením obnovit i předchozí záznamové vrstvy. To je dáno tím, že magnetický záznam bitů zapsaných na stejnou pozici se zcela nepřekrývá. Za tímto účelem byl vytvořen nástroj shred, který opakovaně přepíše zvolený soubor různými vzorci tak, aby nebylo možné data obnovit.

Jelikož v unixových systémech by mělo platit, že vše je soubor, lze tento nástroj použít i pro jakékoliv blokové zařízení. Problém může nastat u normálních souborů uložených na žurnálovacích souborových systémech, kde samotný žurnál může průběh této operace narušit. Dlužno dodat, že na souborových systémech, které nepoužívají plné žurnálování, ale pouze žurnálování metadat, by s touto operací problém být neměl.

## Zálohy a bezpečnost šifrování

Ti, kdo podstupují celé martyrium se šifrováním, tak obvykle činí proto, že raději o data přijdou, než aby se dostala do nepovolaných rukou. Samotné šifrování komplikuje záchranné operace a zapomenutím hesla nebo zničením či poškozením média se šifrovacím klíčem dojde de facto ke ztrátě dat. Nehledě na to, že samotný dm-crypt a LUKS jsou software jako každý jiný, a chyba v nich nebo chyba jinde v jádře být může.

I data na šifrovaných svazcích je možné zálohovat. Ale samotné zálohování citlivých dat je problém - má-li šifrování smysl, musí být adekvátně zabezpečené i zálohy, tj. buď také šifrované, nebo bezpečně uložené.

## Bezpečnost šifrování

Šifrování má mnoho úskalí a slabých míst. Vždy samozřejmě záleží na tom, jaká data a před kým/čím je chráníte. Z toho vám pak vyplynou možní útočníci i jejich znalosti či možnosti. Případné útoky mohou být náhodné (zloděj odnese něčí laptop za účelem prodeje hardwaru) nebo cílené (útočník si počká, až necháte zapnutý laptop s přihlášeným uživatelem a přístupem k citlivým datům bez dozoru a v té chvíli vám ho sebere nebo si data rychle vytáhne).

Šifrování dává jeho uživatelům jistou iluzi absolutního bezpečí - data jsou šifrována nejmodernější a nejsilnější šifrou, dlouhým a komplexním klíčem, za pomoci špičkového softwaru. Problémem je, že samotné šifrování není synonymem pro fyzické zabezpečení dat, je pouze jedním z jeho nástrojů. V následujících bodech se vám pokusím nastínit některé možné útoky na diskové šifrování a možnosti prevence, pokud nějaké jsou:

- útok na nešifrovaný /boot - útočník může modifikovat iniciální ramdisk, aby ukládal zadané heslo, a poté si jej vyzvednout; obrana - nosit /boot na flash disku a zavádět systém z něj, nebo při každém startu ověřovat kontrolní součty
- kamera či útočníkovo oko sledující zadání hesla; obrana - dávat si pozor při zadávání hesla, možné kompromitované heslo revokovat či používat flashdisk s klíčem
- HW keylogger - útočník může nasadit štěnici mezi klávesnici a počítač, čímž odposlechne heslo; obrana - zajistit fyzickou bezpečnost počítače, používat flashdisk s klíčem
- cold boot attack - útočník může získat heslo přímo z RAM, má-li fyzický přístup k počítači; obrana - zajistit fyzickou bezpečnost počítače, vypínat nebo hibernovat počítač když ho není třeba, nepoužívat suspend u laptopů, pokud v rámci suspendu nedojde k výmazu hesel z paměti
- fyzický útok či nátlak na znalce hesla - viz oblíbený komiks; obrana - záleží na typu útočníka, ideální je používat kromě hesla i klíčový soubor (key file) třeba na flash disku a ten včas zničit, popř. používatpopiratelné šifrování (což LUKS vzhledem k hlavičce rozhodně není)

## Správa linuxového serveru: Úvod do SSH pro správce

Každý administrátor unixových serverů by měl být seznámen se SSH a jeho schopnostmi a možnostmi. V tomto díle se podívám na zoubek SSH jako takovému, proberu široké možnosti jeho použití a některé nástroje, které vám umožní využít SSH pohodlně a bezpečně.

### SSH pro správce

SSH lze z pohledu administrátora chápat jako šifrovaný tunel mezi dvěma počítači na bázi klient/server. Přes něj můžeme přistupovat k shellu vzdáleného počítače, ale také kopírovat soubory, spouštět vzdáleně Xkové aplikace, vytvářet roury z místního počítače na vzdálený či vytvářet šifrované tunely pro vzdálený přístup k nějaké službě, která je za firewallem. Prostřednictvím sshfs je dokonce možné připojit vzdálený souborový systém přes FUSE na místní adresář.

#### *Přehled možností SSH*

Jelikož základy SSH včetně spektra jeho možností probral již miniseriál Michala Vyskočila s názvem *SSH receptem na bezpečnost* - první díl, druhý díl, nebudu se o možnostech SSH příliš rozepisovat, dovolím si pouze v rychlosti projít možnosti SSH s jednoduchými příklady:

Přihlášení se ke vzdálenému shellu (všimněte si parametru -p, který umožňuje zvolit jiný než standardní port 22 na cílovém serveru):

```
ssh -p cislo_portu uzivatel@server
```

Pro provedení jediného příkazu nepotřebujete přístup na vzdálený shell, příkaz můžete volat přímo. Příkladem může být získání výpisu obsazeného místa na připojených souborových systémech na serveru, který má SSH server na nestandardním portu, a sice 8022:

```
ssh -p 8022 uzivatel@server "df -h"
```

Prostřednictvím SSH můžete vytvořit rouru začínající na jednom počítači a končící na jiném:

```
mistni_program | ssh -p cislo_portu uzivatel@server 'cat > vdaleny_soubor.dat'
```

Výše zmíněný příkaz pošle výstup místního programu přes SSH tunel vzdálenému programu cat, který data uloží do souboru. Použít můžete samozřejmě kterýkoliv program schopný přijímat data ze standardního vstupu. Tímhle způsobem se dají snadno provádět rychlé zálohy:

```
tar cf - /adresar | xz | ssh uzivatel@server "dd of=zaloha.tar.xz"
```

Dají se tak také zálohovat celé diskové oddíly:

```
dd if=/dev/disk | xz | ssh uzivatel@server "dd of=zaloha_disk.img.xz"
```

Dodávám, že toto se hodí výlučně v případě okamžité potřeby, na pravidelné zálohování máme jiné nástroje. Stejně tak je jasné, že zálohování tohoto rázu by se nemělo provádět na systému s běžícími službami.

SSH dokáže také vytvářet šifrované tunely pro TCP/IP komunikaci. Je tedy možné si na port místního počítače namapovat vzdálenou službu, která je třeba za firewallem, jako v případě vzdálené PostgreSQL databáze v tomto příkladu:

```
ssh -p ssh_port_firewallu -N -f uzivatel@firewall -L 5000:server:5432
```

V tomto případě se asi hodí menší komentář - SSH vytvoří šifrovaný tunel mezi místním počítačem a firewallem, který provozuje SSH server na portu ssh_port_firewallu. Tento tunel bude směřovat na port 5432 počítače s názvem server. Tento vzdálený port pak bude k dispozici jako služba běžící na portu 5000 místního počítače. Tudíž, pokud se po provedení tohoto příkazu spojím s portem 5000 na místním počítači, požadavek se přenese šifrovaným tunelem na firewall, a ten pak požadavek předá (pozor, již nešifrovaně!) serveru, jehož odpověď pak pošle zpět. Parametr -N signalizuje, že se nebude na serveru spouštět žádný příkaz, jen se vytvoří tunel. Parametr -f zařídí, že se tunel po přihlášení vytvoří na pozadí.

Pokud by byl server přístupný přímo, a PostgreSQL na něm běžel jako místní služba (tj. na localhostu), postačil by následující příkaz:

```
ssh -N -f uzivatel@server -L 5000:127.0.0.1:5432
```

SSH umí vytvářet i "reverzní" tunely, pomocí kterých můžete naopak zpřístupnit místní službu na vzdáleném počítači:

```
ssh -N -f uzivatel@server -R 5000:localhost:22
```

Tento příkaz vytvoří šifrovaný tunel mezi vaším počítačem a serverem, který zpřístupní váš místní port 22 (kde obvykle běží SSH) klientům serveru na portu 5000. Tudíž, pokud se nějaký klient spojí s portem 5000 na serveru, bude přes tunel přistupovat k vašemu místnímu portu 22.

Přes SSH je možné pouštět i aplikace využívající grafického rozhraní, je-li to na serveru povoleno (volbaX11Forwarding). Pro každý vzdálený server si můžete vytvořit specifickou konfiguraci v souboru .ssh/configve svém domovském adresáři, kde můžete specifikovat i konkrétní SSH klíč, který se má k přístupu k serveru použít. Pro správu serverů přes nespolehlivé spojení se vám může hodit program screen, o kterém se dozvíte vše potřebné v tomto článku.

#### *sshfs*

Sshfs je FUSE modul, který umožňuje připojovat vzdálené souborové systémy prostřednictvím SSH/SCP:

```
sshfs uzivatel@server:/cesta /kam/pripojit
```

Souborový systém sshfs má celou řadu konfiguračních voleb, které se dozvíte z manuálové stránky. Já si dovolím upozornit na jednu věc, se kterou jsem se setkal, a sice problém s připojením Git repozitáře přessshfs. Pokud se provede připojení s výchozím nastavením, Git odmítá poslušnost. Tento problém lze vyřešit následující volbou:

```
sshfs -o workaround=rename uzivatel@server:/cesta /kam/pripojit
```

Na tomto příkladu je patrné, že sshfs není úplně bezproblémová záležitost a neměl by být brán jako ekvivalent lokálně připojeného souborového systému.

#### *fish*

V KDE aplikacích je možné využít pro přístup k vzdálenému souborovému systému protokol "fish", který umí pohodlně připojit vzdálený souborový systém přes SSH, a to prostřednictvím URL ve tvarufish://uzivatel@server.

#### *SSH servery*

Nejznámějším SSH serverem je OpenSSH, vyvinut původně jako součást unixového operačního systému OpenBSD. Tento nástroj převzaly díky jeho licenci prakticky všechny linuxové distribuce. OpenSSH ale není jediným SSH serverem, těch je dokonce celá řada. Kromě komerčních a proprietárních variant máme pro GNU/Linux k dispozici Dropbear, který vyniká především svou lehkostí, pro kterou se výborně hodí jak na embedded zařízení, tak třeba na virtuální servery s hodně omezenou pamětí (obvykle ty levnější tarify), kde vám ušetří nějaký kus paměti. Na ostatní SSH servery se můžete podívat sem. Dodávám, že některé funkce SSH, které jsem tu popsal, nemusí být k dispozici ve všech SSH serverech.

#### *SSH klienti*

SSH klientů je také celá řada. Kromě klasického ssh klienta z balíku OpenSSH vás upozorním na grafického klienta Putty, který je velmi známý zejména v prostředí MS Windows. Umí toho celou řadu a nehodí se ani zdaleka jenom k přihlašování přes SSH. Podpora SSH klíčů je samozřejmostí, i když klíče je třeba před použitím konvertovat (Putty nepodporuje klasické klíče generované v rámci OpenSSH, tedy ne přímo). Putty funguje dokonce i nativně v Linuxu, i když v Linuxu nám stačí emulátor terminálu a řádkový klient. Něco málo o klíčích v Putty se dozvíte zde.

Pro kopírování souborů (scp/sftp) je možné použít řádkového klienta scp, výše zmíněného "protokolu" fish v KDE aplikacích nebo sshfs. V prostředí MS Windows lze použít grafický nástroj WinSCP, který se podobá klasickému dvoupanelovému diskovému manažeru.

### SSH klíče

Toto téma je z teoretického i praktického hlediska rozebráno v tomto článku, já si dovolím toto téma v rychlosti prolétnout a připravit pro vás opět menší kuchařku.

SSH klíče fungují na bázi asymetrického šifrování, podobně jako GnuPG a jiné aplikace. Vygenerujete si pár klíčů, jeden soukromý (chráněný heslem, obvykle $HOME/.ssh/id_rsa) a jeden veřejný (obvykle$HOME/.ssh/id_rsa.pub). Veřejný klíč nahrajete do souboru .ssh/authorized_keys v domovském adresáři vzdáleného uživatelského účtu, a poté se můžete na server hlásit pomocí svého soukromého SSH klíče. Soukromý klíč není nutné chránit heslem, avšak pak je jasné, že případné zcizení souboru s klíčem může útočník přistupovat ke všem účtům na serverech, kde je umístěn veřejný klíč v authorized_keys. Chránění soukromého klíče heslem je proto vhodné bezpečnostní opatření.

Vygenerování páru klíčů provedete příkazem:

```
ssh-keygen
```

Po vygenerování páru klíčů máte svůj soukromý klíč k dispozici v .ssh/id_rsa (pokud jste se nerozhodli jej vygenerovat jinam) - tento klíč byste měli střežit jako oko v hlavě. Naopak veřejný klíč, který máte k dispozici v .ssh/id_rsa.pub, můžete nahrávat do souboru .ssh/authorized_keys uživatelských účtů na vzdálených serverech, kam se chcete přihlašovat:

```
scp $HOME/.ssh/id_rsa.pub uzivatel@server:
ssh uzivatel@server
uzivatel@server$ mkdir .ssh
uzivatel@server$ cat id_rsa.pub >> .ssh/authorized_keys
```

V authorized_keys může být veřejných klíčů více, proto je vhodné používat přesměrování vstupu v režimu append (dvě větší za sebou). Klíč lze pak "revokovat" prostým výmazem odpovídajícího řádku v tomto souboru.
Pokud máte více soukromých klíčů na různé servery, můžete buď využít ssh agent popsaný dále, nebo jednoznačně specifikovat soukromý klíč, který chcete použít při připojení k serveru (výchozí je~/.ssh/id_rsa):

```
ssh -i .ssh/soukromy_klic uzivatel@server
```

***SSH Agent a Keychain***

Pokud se přihlašujete na servery přes SSH často, popřípadě chcete své klíče chránit heslem, ale současně nechcete být obtěžováni neustálým zadáváním hesla k danému klíči, můžete využít nástroj ssh-agent, který si jedno nebo více hesel ke klíčům zapamatuje, a pak už se vás ssh klient na hesla ptát nebude. Více informací o ssh-agentovi se dozvíte v tomto článku.
Já si dovolím doplnit informaci o nástroji keychain, který vám může usnadnit práci se ssh-agentem. V jeho případě postačí do .bashrc přidat následující dva řádky:

```
/usr/bin/keychain $HOME/.ssh/id_rsa
source $HOME/.keychain/$HOSTNAME-sh
```

Tím si zajistíte, že si Keychain vyžádá hesla ke klíčům, o které se má starat, a podle potřeby spustí ssh-agenta nebo jej pouze přidá do proměnných prostředí, pokud už běží. To vše provede po spuštění shellu. Každá další instance shellu (třeba v rámci virtuálního terminálu) se pak připojí k už spuštěnému ssh-agentovi, bez nutnosti opět zadávat heslo. Keychain má řadu voleb, které můžete využít. Já používám tyto (potlačují zbytečné výpisy a ignorují proměnnou SSH_ASKPASS):

```
/usr/bin/keychain -Q -q --nogui ~/.ssh/id_rsa
```

**Závěr**

SSH je velmi mocný nástroj, který umí mnohé. Proto je vhodné vnímat SSH z pohledu správce nikoliv jako nástroj pro vzdálenou práci se shellem, ale jako komplexní nástroj umožňující vytvářet šifrované, zabezpečené tunely mezi dvěma nebo více počítači, a posílat přes ně nejrůznější data a přistupovat k nejrůznějším službám. SSH je samozřejmě pouze nástroj, jehož faktická bezpečnost záleží na správcích i uživatelích. V příštím díle se podívám právě na bezpečnost SSH, zejména pak z pohledu správců serverů.

## Správa linuxového serveru: Praktické rady pro zabezpečení SSH

SSH je mocný nástroj a SSH server je obvykle také první věc, kterou správce na server instaluje. Jak jej ale zabezpečit proti častým útokům nebo proti případnému zneužití ze strany uživatelů serveru?

### Proti čemu se bránit

Útoky na SSH můžeme rozdělit do dvou kategorií - útoky *vnější* a útoky *vnitřní*. Vnějším útokem je útok neoprávněného uživatele, někoho, kdo na serveru SSH účet nemá. Vnitřním útokem je útok oprávněného uživatele, který přístup k SSH má. Cílem obrany proti vnějšímu útoku je, aby útočník nebyl schopen získat přístup na server, ať již zkoušením často používaných slovníkových hesel nebo využitím nějaké zranitelnosti v SSH. Cílem obrany proti vnitřnímu útočníkovi je nastavit oprávnění tak, aby byl daný uživatelský účet schopen provádět vše, co uživatel požaduje, ale současně aby minimalizoval dopad případného útoku (ten útok samozřejmě nemusí být iniciován daným uživatelem, ale i útočníkem, který zcizí jeho přihlašovací údaje nebo se mu je podaří uhádnout). V tomto dílu "nakousnu" převážně zabezpečení SSH serveru proti vnějšímu útočníkovi.

### Útoky na SSH

Nejčastější útoky na SSH provádí automatizované nástroje, které prohledávají určitý rozsah IP adres, hledají SSH server (většinou pouze na standardním portu) a pokud ho najdou, zkouší nejtypičtější kombinace uživatelských jmen a hesel ve snaze získat přístup přes nezabezpečený účet. Pokud SSH server nenajdou, tak jdou o dům dál. Tuto aktivitu poznáte velice snadno nahlédnutím do logu:

```
auth.log:May 10 14:14:22 hyperion sshd[8182]: Invalid user test from 218.56.61.114
auth.log:May 10 14:14:27 hyperion sshd[8216]: Invalid user guest from 218.56.61.114
auth.log:May 10 14:14:31 hyperion sshd[8254]: Invalid user admin from 218.56.61.114
auth.log:May 10 14:14:40 hyperion sshd[8328]: Invalid user admin from 218.56.61.114
auth.log:May 10 14:14:44 hyperion sshd[8362]: Invalid user user from 218.56.61.114
auth.log:May 10 14:15:04 hyperion sshd[8530]: Invalid user test from 218.56.61.114
```

Tento typ útoků je na provedení velice triviální a je momentálně typem, se kterým se zcela jistě setkáte, pokud neomezíte přístup na svůj SSH port. Obrana bývá triviální - jedna útočící IP adresa a mnoho neúspěšných pokusů za jednotku času, to lze odfiltrovat velmi dobře (nástroje pro tento účel představím níže).

Někteří útočníci jsou ale chytřejší - útoky jsou pak vleklejší (doba mezi pokusy je podstatně delší), popřípadě ještě navíc pocházejí z mnoha různých IP adres (nejčastěji z nějakého botnetu). Sofistikovanější útočníci pak nezkoušejí kombinace jmen a hesel, ale hledají starší, neaktualizované SSH servery, u kterých mohou využít dostupných exploitů objevených zranitelností.

Nejhorším typem útoku je útočník, který si opatří velký seznam serverů se SSH a čeká na 0-day SSH exploit, který až se objeví, celý seznam rychle projede. Proti tomu se brání těžko (leda citelně omezit přístup k SSH). Mimochodem, zálohujete, že?

### Elementární zabezpečení

Zkontrolujte si, že v konfiguračním souboru SSH serveru (pro OpenSSH je to nejspíše/etc/ssh/sshd_config) je povolen pouze protokol verze 2, verzi 1, která má řadu bezpečnostních zranitelností, byste měli mít zakázanou:

```
Protocol 2
```

Z bezpečnostních důvodů je vhodné využívat tzv. striktní režim, který zkontroluje práva v domovském adresáři uživatele, zdali do kritických míst není umožněn přístup jiným uživatelům (a pokud ano, přihlášení odmítne):

```
StrictModes yes
```

Pokud nepotřebujete spouštět na serveru Xkové aplikace, rozhodně zakažte X11Forwarding:

```
X11Forwarding no
```

Pokud nechcete, nehodláte nebo nepotřebujete využívat tunelování v rámci SSH, raději jej zakažte:

```
PermitTunnel no
```

To platí dvojnásob pro servery, kde se k SSH přihlašují běžní uživatelé a ne pouze administrátoři - příslušní uživatelé by pak mohli server využít jako proxy, třeba k anonymnímu surfování, stahování torrentů, apod.

### Elementární řízení přístupu uživatelů k SSH serveru

Předně je třeba zvážit, kdo skutečně potřebuje SSH účet. Je skutečně potřeba, aby každý uživatel, který má třeba na serveru poštovní schránku, měl současně i SSH účet s možností přístupu k shellu? Potřebuje zákazník webhostingu SSH konto? Je nutné, aby správce serveru měl možnost k němu přistupovat z libovolné IP adresy? Zabezpečení SSH serveru by určitě mělo začít analýzou, kdo skutečně potřebuje přístup k SSH a odkud.

#### *AllowUsers, AllowGroups*

Já osobně řeším na svých serverech tento problém tak, že vytvořím skupinu, třeba sshusers, kterou přiřadím jako parametr volby AllowGroups v konfiguračním souboru SSH serveru a do které zařadím všechny uživatele, kteří mají mít přístup k SSH. Pokud je těchto uživatelů velmi málo, není třeba to řešit přes skupiny, postačí jednotlivé uživatele vypsat, oddělené mezerou, jako parametr volby AllowUsers. Jakmile v konfiguraci SSH serveru použijete některou z těchto voleb, nikdo jiný než specifikovaní uživatelé či skupiny se k SSH již nepřihlásí. Pro názornost ilustruji obě varianty pro dva uživatele:

```
/etc/ssh/sshd_config:
AllowGroups sshusers

bash ~# useradd -G sshusers michal
bash ~# useradd -G sshusers honza
```

A druhá varianta:

```
/etc/ssh/sshd_config:
AllowUsers michal honza
```

OpenSSH má k dispozici adekvátní protějšky k oběma volbám - DenyUsers a DenyGroups, které fungují přesně opačně (zabrání daným uživatelům a skupinám přihlásit se k SSH). OpenSSH prochází tyto volby v rámci řízení přístupu v následujícím pořadí:

- DenyUsers
- AllowUsers
- DenyGroups
- AllowGroups

#### *PermitRootLogin*

Další zajímavou volbou je PermitRootLogin, která určuje, zdali se k SSH smí přihlásit uživatel root. Obvykle se doporučuje vytvořit jiného, neprivilegovaného uživatele, a administrátorské úkony řešit přes sudo, popřípadě su. Je to dáno jednak všemocností uživatele root, a jednak tím, že v případě uživatelského účtu se musí případný útočník strefit jak do hesla, tak do uživatelského jména. V případě uživatele rootpotřebuje útočník pouze jediný údaj - heslo.

Volba PermitRootLogin má tři možné hodnoty, yes, no a without-password. Nejzajímavější je právě ona třetí hodnota, without-password, která způsobí, že se uživatel root nebude moci přihlásit heslem, ale pouze některou z jiných metod (tj. třeba SSH klíčem).

#### *PermitEmptyPasswords*

Pokud používáte ověřování heslem, rozhodně se vyplatí vypnutí přihlašování u účtů bez hesla, aby případnému útočníkovi nestačilo pouze uhádnout heslo:

```
PermitEmptyPasswords no
```

#### *Vypnutí ověřování přístupu heslem*

Pokud všichni administrátoři, popřípadě uživatelé, využívají k přístupu na server SSH klíče, pak není důvod, proč povolovat autentikaci heslem. Tu je pak možné z bezpečnostních důvodů vypnout úplně:

```
PasswordAuthentication no
```

To je vůbec jeden z nejideálnějších způsobů zabezpečení SSH, protože pak veškeré pokusy o uhádnutí hesla skončí podobně jako tento:

```
# ssh uzivatel@magnetar.cz
Permission denied (publickey).
```

Útočník ani nedostane příležitost zadat heslo, natož jej uhádnout. Samozřejmě byste měli příslušné SSH klíče střežit jako oko v hlavě a mít na paměti, že pokud příslušné SSH klíč(e) ztratíte, ztratíte i možnost přihlášení na server, protože heslem se tam už pak nedostanete.

## Elementární řízení přístupu k SSH serveru

### *ListenAddress*

Nechte SSH server naslouchat jen tam, kde je to třeba. Máte-li někde kupříkladu firewall, a nechcete se k jeho SSH serveru připojovat zvenčí, pak upravte volbu ListenAddress tak, aby SSH server poslouchal jen tam, kde potřebujete, třeba na nějaké adrese místní sítě (tato IP adresa musí být přidělena konkrétnímu síťovému rozhraní firewallu):

```
ListenAddress 10.0.1.15
```

### *Přesun SSH portu*

Toto je mezi správci velmi kontroverzní záležitost. Vždy, když uvedete přesměrování SSH portu jako bezpečnostní opatření, rozdělí se přítomní správci do dvou nesmiřitelných táborů, kde jeden tuto metodu za zabezpečení považuje, zatímco druhý nikoliv. Já patřím k těm, kteří neuznávají vliv tohoto opatření na zabezpečení serveru a myslím si, že ani vy byste se na toto opatření neměli dívat jako na opatření jakkoliv zvyšující bezpečnost (získali byste jen falešný pocit bezpečí, který je tím nejhorším, co se vám při zabezpečování serveru může stát).

Jediný přínos tohoto opatření spočívá v tom, že se vám vyhne většina automatizovaných útoků, které testují pouze port 22 na přítomnost SSH serveru. Problémem je, že automatizované útoky tohoto rázu jsou ty nejméně nebezpečné, a vaše konfigurace by měla SSH server před těmito útoky spolehlivě zabezpečit, a to zcela bez ohledu na toto opatření. Navíc inteligentní útočník může váš server proskenovat a port objevit, popřípadě může daný port objevit naslouchánímna nějakém uzlu mezi vaším serverem a klientem. Nicméně, pokud máte dobrý důvod toto opatření nasadit, pomůže vám volba Port:

```
Port 61582
```

### *hosts.allow a hosts.deny*

Kromě firewallu je možné omezit přístup k SSH prostřednictvím souborů /etc/hosts.allow a/etc/hosts.deny. Tyto dva soubory řídí přístup k řadě služeb využívajících TCP Wrapper metodu řízení síťového přístupu. Tato metoda spočívá v tom, že daný démon využívá jisté knihovny (libwrap), která pak samotné řízení přístupu provádí. Pokud provozujete síťové služby, které tuto knihovnu nevyužívají, pak je touto metodou přirozeně neochráníte, a nezbyde vám než je chránit jinak (třeba firewallem).

Více informací naleznete v manuálových stránkách příslušných souborů. Základy práce s nimi jsou následující. V souboru hosts.allow se nastavuje povolení přístupu k jednotlivým službám, zatímco v souboru hosts.deny se nastavuje zamezení přístupu ke službám. Při řízení přístupu se upřednostňujehosts.allow, tj. pokud v tomto souboru nějaký počítač nebo síť povolíte, pak se přístup povolí bez ohledu na obsah hosts.deny. Jako příklad uvedu následující situaci:

```
/etc/hosts.allow:
ALL: localhost
ALL: 10.0.5.15
sshd: 12.34.56.78

/etc/hosts.deny:
ALL: 1.2.3.4
sshd: 5.6.7.8
```

V tomto příkladu bude jednoznačně povolen přístup ke všem službám z localhostu a 10.0.5.15 a k SSH z IP adresy 12.34.56.78. Zakázán bude přístup ke všem službám adrese 1.2.3.4 a jen k SSH adrese 5.6.7.8. Je samozřejmě možné specifikovat rozsahy i různé zástupné výrazy (jako třeba ALL. Více informací se dozvíte v manuálových stránkách.

## Pokročilé řízení přístupu uživatelů k SSH serveru

### *access.conf*

V tuto chvíli víte, jak omezit přístup k SSH, ale co když budete potřebovat omezit přístup k SSH třeba jen pro některé uživatele s tím, že u jiných uživatelů vám třeba nevadí, odkud se hlásí? V takovém případě byste se měli podívat blíže do souboru /etc/security/access.conf, kde je možné specifikovat, kteří uživatelé se smí přihlašovat z jakých IP adres nebo sítí. Ukažme si to na příkladu:

```
+ : root : 127.0.0.1
+ : root : 10.0.0.0/8
+ : root : 12.34.56.78
- : root : ALL
+ : michal : example.com
+ : michal : 1.2.3.4
+ : @admini : 5.6.7.8
- : ALL : ALL
```

Plus nebo mínus na začátku udává, zdali se má přístup povolit nebo zakázat. Oddělovačem je dvojtečka, následuje uživatel či skupina (skupina je uvozena zavináčem) a posledním parametrem je síť, doména nebo IP adresa, která je předmětem řízení přístupu. V prvních třech řádkách je povolen přístup uživateli root z localhostu, místního rozsahu 10.0.0.0/8 a vnější IP adresy 12.34.56.78. Na čtvrtém řádku je pravidlo, které zamezí přístup k SSH uživateli root z jakéhokoliv jiného umístění. Následuje povolení přístupu uživatelimichal z domény example.com a IP adresy 1.2.3.4. Předposlední řádek povoluje přístup skupině admini z IP adresy 5.6.7.8. Úplně poslední řádek pak funguje jako "vykopávač", zamezí jakýmkoliv jiným přístupům odjinud. Více informací viz manuálová stránka souboru access.conf

## Správa linuxového serveru: Praktické rady pro zabezpečení SSH II

Minule jsem probral některé metody zabezpečení SSH, dnes v tom budu pokračovat. Proberu elementární zabezpečení SSH firewallem a představím nástroj Denyhosts.

### Pravidelná aktualizace

Začnu velmi zlehka něčím, co se sice obvykle považuje za naprostou samozřejmost, a sice pravidelnou aktualizací a sledováním hlášení o bezpečnostních problémech. Bohužel se i na takovou samozřejmost v některých případech zapomíná, což mívá za následek úspěšný průnik útočníka, který se dříve či později skoro jistě dostaví. V případě distribuce Debian naleznete základní informace o bezpečnosti na stránkách distribuce. Hodit se vám může také e-mailová konference, do které se můžete přihlásit, abyste dostávali informace o bezpečnostních aktualizacích a problémech.

S aktualizacemi souvisí i vývojový cyklus distribuce, kterou používáte na serveru. Pokud se jedná o distribuci s pravidelným vývojovým cyklem, jako třeba Debian (resp. jeho stable větev), pak vás během používání určitého vydání čeká řada drobných aktualizací, v rámci kterých se obvykle nemění verze aktualizovaných komponent. Takové aktualizace by neměly narušit fungování vašich služeb (snad s výjimkou restartu při aktualizaci), i když výjimky potvrzující pravidlo samozřejmě existují. Jednou za nějaké období vás pak čeká velká aktualizace, kde se poměrně dramaticky mění verze komponent, díky čemuž může dojít k narušení fungování některých komponent či démonů (došlo ke změnám v konfiguračních souborech či ve funkcionalitě).

Některé aktualizace během života konkrétního vydání mohou vyžadovat nějaký zásah správce (namátkouprůšvih Debianu s OpenSSL knihovnou), takže i když je teoreticky možné provádět aktualizace automaticky cronem, nelze tento postup doporučit, alespoň ne na produkčních serverech. Abyste věděli, kdy máte na serveru k dispozici nové aktualizace, můžete použít v případě Debianu nástroj Apticron, o kterém se dočtete více v článku na debian-administration.org. Funguje tak, že jednou denně zjistí dostupné aktualizace a pošle vám o nich e-mail, pokud nějaké k dispozici jsou. Vy se pak můžete rozhodnout, zdali aktualizaci provedete nebo ne.

Upgrade na novější verzi distribuce je vhodné dělat opatrně a s rozmyslem, ideálně až nějakou dobu po vypuštění nové verze, po pečlivém prostudování poznámek k vydání (release notes), kde se obvykle nachází většina známých problémů, na které můžete při upgradu narazit. Bývá dobré si připravit půdu pro případný rollback provedením kompletní zálohy dat i konfigurace, a nebo ještě lépe, provést zkušební upgrade na jiném počítači se stejnou konfigurací (konfiguraci i data z produkčního serveru můžete na zkušební server zkopírovat).

### Elementární zabezpečení SSH firewallem

Provozu firewallu v Linuxu se budu blíže věnovat v některém z příštích dílů. V tuto chvíli pro potřeby zabezpečení SSH postačí následující jednoduchá sada pravidel. V tomto příkladu nevyužívám možnosti stavového firewallu, což příklad trošku zjednoduší, a hlavně to bude fungovat i na virtuálních serverech, kde není k dispozici příslušný modul pro stavový firewall:

```
# povolení přístupu k SSH z IP adresy 1.2.3.4
iptables -A INPUT -s 1.2.3.4 -p tcp --dport 22 -j ACCEPT
# zamezení přístupu k SSH z IP adresy 5.4.3.2
iptables -A INPUT -s 5.4.3.2 -p tcp --dport 22 -j DROP
# zamezení přístupu k SSH z rozsahu adres 10.0.0.0/8
iptables -A INPUT -s 10.0.0.0/8 -p tcp --dport 22 -j DROP
```

Akci DROP můžete pochopitelně vyměnit za REJECT s příslušnou ICMP zprávou, pokud chcete. V případě, že potřebujete řídit přístup k SSH třeba na routeru, kde je více síťových rozhraní, můžete být specifičtější a vzít dané rozhraní v úvahu:

```
# zákaz přístupu na SSH z rozhraní eth0
iptables -A INPUT -i eth0 -p tcp --dport 22 -j DROP
# zákaz přístupu na SSH z rozhraní eth1
iptables -A INPUT -i eth1 -p tcp --dport 22 -j DROP
# povolení přístupu k SSH na rozhraní eth2 z IP adresy 10.0.1.5
iptables -A INPUT -i eth2 -s 10.0.1.5 -p tcp --dport 22 -j ACCEPT
# zamezení přístupu k SSH ze všech ostatních adres na rozhraní eth2
iptables -A INPUT -i eth2 -p tcp --dport 22 -j DROP
```

Pro úplnost uvádím trošku elegantnější řešení s použitím stavového firewallu a vlastního řetězce, kde pak můžete zpracovávat pouze žádosti o nové spojení se standardním SSH portem:

```
# vytvoření řetězce pro ssh
iptables -N ssh
# přesměrování paketů s žádostí o vytvoření spojení na SSH port do řetězce ssh
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ssh
# v rámci řetězce SSH povolení paketů z IP adresy 1.2.3.4
iptables -A ssh -s 1.2.3.4 -j ACCEPT
# v rámci řetězce SSH povolení paketů z rozsahu 10.0.0.0/8
iptables -A ssh -s 10.0.0.0/8 -j ACCEPT
# zahození ostatních paketů
iptables -A ssh -j DROP

# někde jinde pak musí být řádka propouštějící pakety náležející již vytvořeným spojením:
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Dodám ještě, že nemusíte využívat Netfilter (a nástroj iptables) přímo, nýbrž můžete použít některou z nadstaveb jako třeba Shorewall, o kterém zde vyšel dvoudílný seriál (první díl, druhý díl).

### Pasivní obrana vs. aktivní obrana

Do této chvíle byly probrány především prvky pasivní ochrany SSH serveru. To jsou bez zásahu správce neměnná opatření, např. přístup k SSH pouze z IP určité adresy nebo naopak zamezení přístupu k SSH z určité IP adresy nebo rozsahu IP adres. Aktivní opatření jsou naopak taková, která reagují na vzniklou situaci, kupříkladu na sadu neúspěšných pokusů o přihlášení zablokováním dané IP adresy.

Výhodou takových opatření je právě možnost rychlé reakce na nějakou vzniklou situaci (zejména pak neúspěšný pokus o přihlášení). Nevýhody těchto opatření vyplývají především z faktu, že tato reakce na vzniklou situaci je sice předem definovaná, ale automatizovaná a nepodléhá schválení správcem. Mezi potenciální problémy aktivních opatření patří možnost zablokování sebe sama, popřípadě jejich využití útočníkem, ať již k DoS útokům nebo, v horším případě, k průniku do systému.

#### *Parsery logů a jejich zranitelnosti*

Nejtypičtější nástroje aktivní obrany jsou parsery logů (např. Denyhosts a Fail2ban). Ty monitorují záznamy v příslušném logu a vyhledávají záznamy jistého typu. Jakmile se takové záznamy objeví, jsou prohnány parserem (nebo spíše nějakým regulárním výrazem) a jsou z nich získány požadované informace. Na jejich základě je pak vyprodukována nějaká reakce (obvykle shellový příkaz nebo přidání záznamu do souboru). Princip je to relativně jednoduchý, avšak skýtá jistou potenciální zranitelnost - *log injection*. Stejně jako je možné využít nekorektně ošetřený SQL vstup ke změně prováděného SQL příkazu, je možné podobným způsobem upravit akci prováděnou daným nástrojem prostřednictvím vhodně "upraveného" záznamu v logu. Dodám, že známé zranitelnosti tohoto typu byly již z oněch dvou zmíněných nástrojů odstraněny a příslušné regulární výrazy byly podstatně zpřesněny. Více informací o těchto zranitelnostech naleznete na ossec.net.

Pokud budete nasazovat vlastní parser, popřípadě budete upravovat funkci stávajícího vlastním regulárním výrazem, určitě mějte na paměti tuto možnost útoku spolu s faktem, že záznamy v logu může v Linuxu běžně generovat jakýkoliv program, včetně interpretru PHP (pokud to samozřejmě správce nezakáže).

#### *Pozor na DoS*

Ať už budete nasazovat jakýkoliv nástroj aktivní obrany, vždy zůstane ten nejjednodušší možný vektor útoku - *Denial of Service*. Útočník se možná k SSH nedostane, ale může za určitých okolností znemožnit vzdálený přístup i správci či například zákazníkům (kteří využívají SSH/SCP ke kopírování souborů na server apod.). Ze způsobů provedení přichází v úvahu jak možné falšování zdrojové IP adresy, tak *log injection* či prosté sdílení IP adresy nebo jejího rozsahu s útočníkem nebo počítačem pod jeho kontrolou (viz botnety). Vzhledem k tomu, že dostupných IPv4 adres ubývá a řada poskytovatelů používá NAT, jsou možné následky tohoto typu útoku citelnější.

## Denyhosts

Denyhosts je parser logů, který hlídá neúspěšné pokusy o přihlášení k SSH a dle nastavení po určitém počtu neúspěšných pokusů zařadí příslušnou IP adresu do /etc/hosts.deny. Využívá TCP Wrapper, tedy jednu z metod probraných v minulém díle. Abyste zabránili možnému odstřihnutí sebe sama při neúspěšných pokusech o přihlášení, zapište příslušné IP adresy nebo rozsahy, ze kterých se na server chcete hlásit, do souboru /etc/hosts.allow. Pokud bude IP adresa současně v obou souborech, přístup z ní se povolí, neboť se při kontrole přístupu postupuje v pořadí hosts.allow a hosts.deny. Tato metoda umožňuje blokovat přístup jak na SSH, tak na všechny služby využívající knihovnu libwrap (u síťových démonů si to můžete ověřit nástrojem ldd).

Výhodou, respektive možná spíše vlastností Denyhosts, je nezávislost na firewallu, který v tomto případě není vůbec využíván. Při práci tohoto nástroje nedochází ke spouštění shellových příkazů s právy roota a parametry získanými regulárním výrazem z logu. Hlavní konfigurační soubor Denyhosts se nachází v/etc/denyhosts.conf. Mezi podstatné volby konfigurace patří následující:

```
BLOCK_SERVICE = ALL
```

Tato část řídí typ záznamu, který se vytvoří v hosts.allow. Hodnota "ALL" způsobí, že se přístup z dané IP adresy zamezí na všechny služby využívající TCP Wrapper na daném serveru. Pokud chcete útočníkům zamezit pouze přístup k SSH, použijte hodnotu "sshd". Primární nastavení chování Denyhosts řídí následující volby:

```
DENY_THRESHOLD_INVALID = 5
DENY_THRESHOLD_VALID = 10
DENY_THRESHOLD_ROOT = 1
```

Toto jsou počty neúspěšných pokusů, které jsou třeba k zablokování dané IP adresy.DENY_THRESHOLD_INVALID udává počet pokusů pro neplatného uživatele, DENY_THRESHOLD_VALID udává počet pokusů u platného uživatele a DENY_THRESHOLD_ROOT udává počet pokusů pro uživatele root. Máte-li podobně "citlivé" účty jako root, u kterých byste chtěli specifikovat jiný počet pokusů než pro ostatní uživatele, podívejte se na dokumentaci k volbě DENY_THRESHOLD_RESTRICTED. Počítadla neúspěšných pokusů se po určité době vynulují. Tuto dobu je možné nastavit následujícími volbami:

```
AGE_RESET_VALID=5d
AGE_RESET_ROOT=25d
AGE_RESET_INVALID=10d
```

Zde uvedené hodnoty by znamenaly, že se počet neúspěšných pokusů pro platného uživatele vynuluje za pět dní, u neplatného uživatele za 10 dní a v případě uživatele root bude muset uběhnout 25 dní. Velmi podstatnou volbou je pak RESET_ON_SUCCESS, který příslušné počítadlo vynuluje při úspěšném přihlášení, je-li nastaven na "yes".

Chcete-li dostávat zprávy o zablokovaných IP adresách e-mailem, zapište svůj e-mail jako hodnotu volbyADMIN_EMAIL. Nezapomeňte zkontrolovat nastavení SMTP (implicitní nastavení využívá místní SMTP server). Upozorňuji, že na jediném serveru dochází obvykle k zablokování několika IP adres denně. Můžete tedy spíše preferovat shrnutí za každý den, o což se vám postará třeba nástroj Logwatch (o tom se v tomto seriálu určitě ještě podrobněji zmíním).

Denyhosts umožňuje výměnu dat o útočnících s centrálním serverem, který spravuje projekt Denyhosts. Chcete-li využívat této možnosti, musíte nejprve odkomentovat řádek s volbou SYNC_SERVER. Můžete specifikovat i jednosměrnou komunikaci (pouze zasílání dat o útočnících či pouze stahování dat o útočnících) pomocí parametrů SYNC_UPLOAD a SYNC_DOWNLOAD. Délku periody pro synchronizaci řídí volbaSYNC_INTERVAL. Je logické předpokládat, že počet záznamů všech uživatelů této služby bude obrovský. Z tohoto důvodu se běžně nestahují všechny IP adresy, ale pouze takové, které útočily na několik serverů (výchozí hodnota je 3). Toto nastavení můžete ovlivnit volbou SYNC_DOWNLOAD_THRESHOLD. Spolu s touto volbou vystupuje ještě jedna volba omezující počet stažených záznamů, a sice SYNC_DOWNLOAD_RESILIENCY. Ta specifikuje, jak "čerstvé" příslušné útoky z daných IP adres jsou - výchozím nastavením je "5h", tedy 5 hodin. Tato volba funguje tak, že se propustí pouze takové IP adresy, kde útočník zaútočil vícekrát během této doby. Tzn. pokud mezi posledními dvěma útoky uběhlo 5 a více hodin, daná IP adresa se stahovat nebude, pokud uběhlo méně, daná IP adresa se stáhne.

Pokud vás v tuto chvíli napadlo, že byste mohli této funkcionality využít centrálně pro své spravované servery, na některém ze svých serverů si zřídit úložiště a to pak využívat k synchronizaci pouze mezi vašimi servery, není to možné - Denyhosts toto v tuto chvíli bohužel neumí.

Tím bych tento díl ukončil. Příště se podívám na konkurenci Denyhosts, projekt Fail2ban, a na některé další zajímavé metody zabezpečení SSH.

## Správa linuxového serveru: Praktické rady pro zabezpečení (nejen) SSH III

V dnešním díle budu v miniseriálu o zabezpečení SSH pokračovat, avšak tentokrát proberu nástroje a metody, které pomohou ochránit i jiné služby než SSH. Představím nástroj Fail2ban a proberu techniku zvanou port knocking.

### Fail2ban

Fail2ban je kolegou v minulém díle představeného nástroje Denyhosts. Zatímco Denyhosts je zaměřen výhradně na ochranu SSH prostřednictvím hosts.deny, Fail2ban je zaměřen obecněji a dokáže ochránit nejenom SSH, ale i řadu dalších služeb. Umožňuje dokonce plné přizpůsobení své činnosti čili není problém si přidat novou hlídanou událost a případnou reakci na ni. Stejně tak je možné přizpůsobit výchozí nastavení.

V Debianu naleznete konfiguraci nástroje Fail2ban v adresáři /etc/fail2ban. Základní nastavení naleznete v konfiguračním souboru jail.conf, nicméně není doporučováno jej modifikovat přímo, ale změny zapsat do souboru /etc/fail2ban/jail.local. Asi to nejdůležitější bude whitelist IP adres, u kterých nechcete přístup omezovat (abyste si server omylem na dálku "nezamkli", nebo to místo vás neprovedl útočník):

```
ignoreip = 127.0.0.1 10.0.0.0/8
```

Jednotlivé IP adresy a rozsahy jsou odděleny mezerou. Zbylé dvě z klíčových voleb jsou bantime, tj. doba, po kterou bude útočící IP adresa zablokována (v sekundách), a maxentry, tj. počet povolených pokusů, po jejichž překročení bude daná IP adresa zablokována:

```
bantime  = 600
maxretry = 3
```

Toto jsou výchozí nastavení, která budou použita, není-li pro danou chráněnou službu specifikována jiná hodnota. Níže v konfiguračním souboru pak naleznete jednotlivé chráněné služby, u kterých jsou tyto parametry obvykle přizpůsobeny (u SSH je ve výchozím nastavení povoleno 6 pokusů o přihlášení). Anatomii nastavení jednotlivých chráněných služeb demonstruji na příkladu:

```
[ssh]

enabled = true
port    = ssh
filter  = sshd
logpath  = /var/log/auth.log
maxretry = 6
```

SSH je u Fail2ban jediná aktivní chráněná služba, ostatní musíte aktivovat úpravou volby enabled, která musí být nastavena na true, má-li služba být aktivní. Dále následuje port, který bude pro danou IP adresu zablokován - ssh označuje port 22, all pak všechny porty. Výchozí akce pro zablokování dané IP adresy zde není specifikována, tudíž je použita výchozí akce iptables-multiport. Pokud se podíváte do adresáře/etc/fail2ban/action.d, naleznete soubor iptables-multiport.conf, který tuto akci řídí. Tato akce zablokuje danou IP adresu pomocí nástroje iptables, vytvořením příslušného pravidla. Za dobu specifikovanou ve volbě bantime toto pravidlo vymaže a z dané IP adresy se může na SSH opět přistupovat.

Fail2ban je parser logů úplně stejně jako nástroj Denyhosts. Volba filter označuje filtr, který se má u dané služby použít, a volba logpath pak specifikuje umístění souboru s příslušným logem. Samotný filtr pak naleznete v adresáři /etc/fail2ban/filter.d ve stejnojmenném souboru s příponou .conf. Filtr je tvořen sadou regulárních výrazů. Pokud budete používat nástroj Fail2ban a budete specifikovat vlastní filtr, buďte opatrní a nezapomeňte na dříve zmiňovanou hrozbu v podobě log injection.

Fail2ban má přednastavená pravidla pro ochranu řady dalších služeb - namátkou Apache, tři nejpoužívanější FTP servery, Postfix a Courier autentikační démon. Ty můžete použít bez nutnosti si příslušná řešení a pravidla nastavit ručně.

Fail2ban démona můžete řídit ručně z příkazové řádky, pomocí nástroje fail2ban-client. Takto si můžete nechat vypsat stav chráněných služeb:

```
debian:~# fail2ban-client status
Status
|- Number of jail:      1
`- Jail list:           ssh
```

Zde vidíte pouze jediný "jail", tedy chráněnou službu, a sice ssh. Stav jednotlivých služeb si můžete nechat vypsat podrobněji:

```
debian:~# fail2ban-client status ssh
Status for the jail: ssh
|- filter
|  |- File list:        /var/log/auth.log
|  |- Currently failed: 1
|  `- Total failed:     94
`- action
   |- Currently banned: 2
|  `- IP list:       77.93.17.145 141.152.17.15
   `- Total banned:     5
```

Zde můžete vidět celkový počet selhaných pokusů o přihlášení (94), počet momentálně blokovaných IP adres (2) i jejich seznam (podotýkám, že IP adresy ve výpisu jsou smyšlené, nejedná se o skutečné útočníky) a celkový počet zablokovaných adres (5).

Pomocí tohoto nástroje je možné činnost Fail2ban precizně řídit z příkazové řádky, jednotlivé chráněné služby je možné odebírat, přidávat či konfigurovat nové. Více se dozvíte v dokumentaci.

### Omezení propustnosti firewallu pro SSH

Jisté zabezpečení SSH může poskytnout modul Netfiltru limit. Tento modul umožňuje omezit počet spojení směřujících na SSH port. Jeho použití je lehce spekulativní, protože neumí brát v úvahu počet spojení z konkrétních IP adres. Přesto může v některých případech pomoci, pokud není k dispozici nic jiného:

```
iptables -N ssh
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ssh
iptables -A ssh -s 1.2.3.4 -j ACCEPT
iptables -A ssh -s 10.0.1.0/24 -m limit --limit 2/sec --limit-burst 20 -j ACCEPT
iptables -A ssh -m limit --limit 5/min --limit-burst 10 -j ACCEPT
iptables -A ssh -j DROP
```

Tato ukázka využívá stavový firewall, tzn. filtrují se pouze žádosti o nová spojení, existující spojení nebudou omezena. Veškerá nová spojení v tomto případě půjdou do řetězce ssh, ve kterém je nejprve povolena IP adresa 1.2.3.4, na kterou se žádná omezení vztahovat nebudou. Dále je specifikován rozsah vnitřní sítě10.0.1.0/24, který je omezen volněji, ostatní pokusy o připojení k SSH jsou pak omezeny přísněji. Poslední pravidlo zahodí jakýkoliv paket, který omezením nevyhoví.

Samotné omezení pomocí modulu limit funguje tak trochu jako revolver, kde --limit-burst představuje počet komor a --limit představuje dobu, za kterou se obnoví jeden náboj. Každý paket, který projde tímto pravidlem, pak "vystřelí" jeden náboj. Pokud dojdou náboje, paket příslušnému pravidlu nevyhoví a pokračuje v řetězu dál. V prvním uvedeném příkladě je počet komor 20 a doba obnovy jednoho náboje 0.5 sekundy (obnoví se dvakrát za sekundu), ve druhém je počet komor 10 a doba obnovy je 12 sekund (obnoví se 5krát za minutu).

Zásadní nevýhody tohoto přístupu jsou dvě. V první řadě, jakýkoliv útok na SSH vyvolá Denial of Service dané služby. Můžete specifikovat whitelist (jako v příkladu výše), takže se pořád budete moci připojit z jistých IP adres, ale pro případné klienty, kteří nejsou ve výjimkách, to znamená nemožnost se připojit. Tento přístup se tedy hodí spíše tam, kde mají k SSH přístup pouze administrátoři.

Druhou nevýhodou je absence přehledu o úspěšnosti pokusu o přihlášení - tato pravidla počítají pouze průchozí pakety bez ohledu na to, zdali patří spojením, která se úspěšně autentikují vůči SSH serveru, nebo spojením, která nepředloží platné autentikační údaje.

### Port knocking

Port knocking je další z lehce kontroverzních záležitostí, alespoň ve své čisté podobě. V zásadě funguje tak, že jistý démon naslouchá přímo na síti (tzn. nezávisle na firewallu), přičemž hledá předem jisté definované sekvence paketů přicházející z nějaké IP adresy. Pokud některou ze sekvencí uvidí, provede nějakou akci, tzn. pustí pod rootem nějaký příkaz a jako parametr mu předá zdrojovu IP adresu daných paketů. Největší problém port knockingu v této podobě spočívá v tom, že útočník naslouchající na síti (ať už blízko na straně serveru nebo blízko na straně klienta) může danou sekvenci odposlechnout a zreprodukovat. Toto se samozřejmě dá vyřešit, a to jedním ze dvou způsobů, které uvedu níže.

***Knockd a fwknop***

Démon knockd je tím nejobvyklejším port knocking démonem. K dispozici spolu s ním je i klient, který umožňuje na porty "klepat". V Debianu je knockd po instalaci neaktivní a čeká na úpravu hlavního konfiguračního souboru /etc/knockd.conf, po kterém můžete démon povolit úpravou /etc/default/knockda spustit. Samotná konfigurace akce vypadá takto:

```
[fwDown]
sequence    = 7001:tcp,8205:tcp,9050:udp
seq_timeout = 5
command     = /etc/init.d/iptables stop
tcpflags    = syn
```

Sekvence portů je poměrně jasně definovaná v položce sequence - očekávají se tři pakety. TCP paket na port 7001 následovaný TCP paketem na port 8205 následovaný UDP paketem na port 9050. TCP pakety pak musí mít příznak SYN (specifikováno v položce tcpflags. Příslušná sekvence musí proběhnout během 5 vteřin (viz položka seq_timeout). Pokud knockd tedy takovou sekvenci paketů objeví, provede příkaz specifikovaný v položce command. Tento naznačený příkaz slouží k zastavení firewallu (třeba pokud byste se omylem "zamkli" při ruční manipulaci s firewallem) - v Debianu ale nebude fungovat, protože Debian nemá ve výchozí instalaci k dispozici skript pro iptables.

Samotná ochrana SSH může vypadat třeba takto:

```
[opencloseSSH]
sequence      = 2022:udp,3303:tcp,4440:udp,5225:udp
seq_timeout   = 2
tcpflags      = syn,ack
start_command = /usr/sbin/iptables -A INPUT -s %IP% -p tcp --syn --dport 22 -j ACCEPT
cmd_timeout   = 5
stop_command  = /usr/sbin/iptables -D INPUT -s %IP% -p tcp --syn --dport 22 -j ACCEPT
```

Tato ukázka je již poněkud šťavnatější. Je zde definována sekvence čtyřech portů, jejíž timeout je nastaven na dvě vteřiny, a jsou specifikovány dva příkazy - start_command a stop_command. První z příkazů otevře SSH port pro danou IP adresu, druhý dané pravidlo z firewallu vymaže. Doba mezi provedením obou příkazů se řídí položkou cmd_timeout, která je v tomto případě nastavena na 5 sekund. Předpokladem fungování této ukázky je kompletní ochrana SSH firewallem - kromě explicitních výjimek musí být port SSH firewallem uzavřený.

Problémem těchto sekvencí je, že se dají poměrně snadno odposlechnout. Když ne někde blízko serveru, tak určitě v rámci sítě, ze které se správce na server hlásí (nezabezpečené bezdrátové sítě tomuto problému v dnešní době dávají mnohem větší rozměr). Je jasné, že k obecnému zvýšení bezpečnosti povede i tento postup - spektrum subjektů, které mohou danou sekvenci odposlechnout, bude podstatně menší než celý Internet klepající na váš otevřený SSH port. Na stranu druhou, dokud je sekvence předem daná a opakuje se, nelze předpokládat, že se k SSH přeci jen nedostane někdo nepovolaný (i když, pak máte samozřejmě stále k dispozici standardní zabezpečovací mechanismy v rámci SSH - včetně nutnosti se na server přihlásit platným jménem a heslem nebo klíčem).

Jedno z řešení tohoto problému umí i knockd, a sice tzv. sekvence na jedno použití (položkaOne_Time_Sequences, jejímž parametrem je soubor s řadou sekvencí). To znamená, že se nedefinuje jedna sekvence, ale celá řada sekvencí, přičemž po použití aktuální sekvence dojde k ukončení její platnosti a server bude očekávat následující sekvenci. Problémem této metody je nutnost synchronizace sekvencí na všech klientech a čas od času nutnost znovu vygenerovat nové sekvence. Bohužel mi není znám jiný způsob, jak si nechat sekvence pro knockd vygenerovat, než si k tomu napsat vlastnískript.

Jinou možností řešit výše uvedený problém je prostřednictvím SPA, tj. single packet authorization, kde se ověření provádí jediným paketem za asistence asymetrického šifrování a GnuPG. Součástí SPA paketů jsou náhodná data, která zajišťují unikátnost každého paketu a neopakovatelnost. Tuto techniku bohužel knockdneumí, umí ji nástroj fwknop, který není k dispozici v repositářích momentální stabilní verze Debianu, nicméně bude k dispozici v příštím stabilním vydání. Tato technika spolu s tímto nástrojem byly popsány podrobněji v jednom z odkazů pod článkem.

## Správa linuxového serveru: Praktické rady pro zabezpečení (nejen) SSH IV

V tomto díle proberu aktivní zabezpečení SSH na úrovni firewallu s využitím modulu recent a nástroj scponly.

### Aktivní zabezpečení SSH na úrovni firewallu

Pokročilejší techniku zabezpečení SSH na úrovni firewallu představuje následující řešení, které tvoří jistou obdobu Denyhosts a Fail2ban, i když stále zde platí, že firewall nemůže zjistit, bylo-li dané přihlášení úspěšné, či nikoliv. Nutností je v tomto případě dostupnost jaderného modulu ipt_recent (v distribučním jádře Debianu tento modul k dispozici je). Příslušná pravidla vypadají zhruba takto:

```
1. iptables -N ssh
2. iptables -A ssh -s 1.2.3.4 -j ACCEPT
3. iptables -A ssh -m recent --update --seconds 15 --hitcount 5 --name SSH -j DROP
4. iptables -A ssh -m recent --set --name SSH -j ACCEPT
5. iptables -A ssh -j ACCEPT

6. iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ssh
```

Pochopit tento příklad je asi trošku obtížnější. Pokusy o připojení k SSH jsou směřovány do řetězce s názvem "ssh" (řádek 6). V tomto řetězci je nejprve povolen přístup z IP adresy 1.2.3.4 (řádek 2). Na řádku 4 je každý procházející paket zařazen do mechanismu modulu recent, a sice do jeho vlastního řetězce s názvem SSH, a propuštěn (tj. povolen). Třetí řádek je pak jádro celého mechanismu. Byla-li daná zdrojová IP adresa procházejícího paketu viděna vícekrát než 5krát za 15 sekund, je zařazena na blacklist a paket je zahozen. Pokud z dané IP adresy nepřijde žádná další žádost o pokus o připojení k SSH během 15 sekund, je daná IP adresa vyjmuta z blacklistu a může se znovu pokusit o přihlášení. Pokud z této IP adresy přijde další žádost, zatímco je v blacklistu, paket je zahozen a počítá se znovu od nuly.

Trošku komplexnější příklad může vypadat takto:

```
iptables -N ssh-blacklist
iptables -A ssh-blacklist -m recent --name blacklist --set
iptables -A ssh-blacklist -m limit --limit 1/minute --limit-burst 100 -j LOG --log-prefix "iptables ssh-blacklist: "
iptables -A ssh-blacklist -j DROP

iptables -N ssh-whitelist
iptables -A ssh-whitelist -s 1.2.3.4 -j ACCEPT
iptables -A ssh-whitelist -j RETURN

iptables -N ssh
iptables -A ssh -j ssh-whitelist
iptables -A ssh -m recent --update --name blacklist --seconds  43200 --hitcount  1 -j DROP
iptables -A ssh -m recent --set     --name short
iptables -A ssh -m recent --set     --name long
iptables -A ssh -m recent --update --name short --seconds   15 --hitcount  5 -j ssh-blacklist
iptables -A ssh -m recent --update --name long --seconds  3600 --hitcount 30 -j ssh-blacklist
iptables -A ssh -j ACCEPT

iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ssh
```

Zde vidíte tři řetězce, řetězec ssh, kam směřují nová spojení s portem 22, dále řetězec ssh-whitelist, kde jsou specifikovány IP adresy a rozsahy, které mají být povoleny bez filtrování (v tomto případě je to jediná adresa - 1.2.3.4). Řetězec ssh-blacklist provádí dvě podstatné úlohy. Jednak zařazuje danou IP adresu na blacklist a příslušný paket zahazuje, ale ještě před tím, než paket zahodí, jej zaloguje. Zde vidíte modullimit, který jsem představil výše. V tomto případě omezuje počet hlášek v logu na průměrně jednu za minutu (s možností najednou pojmout až 100 hlášek). Podstatné je, že do tohoto řetězce se příslušný paket dostane pouze jednou, a sice když je na blacklist zařazen - jinak ho zahazuje první "recent" pravidlo v řetězci ssh. Pro blacklist jsou stanovena dvě různá časová pásma, a sice 5 spojení za 15 sekund a 30 spojení za hodinu (3600 sekund). Pokud se nějaká IP adresa pokusí o více spojení za daný časový úsek, bude zařazena na blacklist. Doba, po kterou pak v blacklistu zůstane, je minimálně 1 den (43200 sekund). Daná IP adresa bude vyjmuta, pokud se během 1 dne od zařazení do blacklistu už nepokusí o přístup k SSH. Pokud to zkusí znovu během této doby, počítadlo se vynuluje a začíná se znovu.

Jak už bylo řečeno, modul recent nebere v úvahu úspěch či neúspěch pokusu o přihlášení, pouze počet průchozích paketů za jednotku času. Oproti modulu limit má však podstanou výhodu v tom, že počítadla paketů jsou specifická pro každou zdrojovou IP adresu průchozího paketu. To znamená, že pokud bude útočník útočit z jedné IP adresy, ta bude brzy zablokována, ale přístup odjinud bude fungovat normálně.

Uznávám, že modul recent je trošku "vyšší dívčí", čemuž bohužel napomáhá i lehký nedostatek podrobnější dokumentace. Popis jednotlivých voleb a základní princip však lze pochopit jednak z výše odkazované dokumentace, a pak z manuálové stránky nástroje iptables.

### scponly

Posledním nástrojem, který v rámci tohoto miniseriálu představím, je scponly. Jelikož SSH a SCP jdou ruku v ruce, může se stát, že v některých případech budete chtít, aby měl některý váš uživatel možnost přes SSH/SCP kopírovat soubory, ale současně aby se nemohl dostat k shellu a pouštět příkazy. A právě k tomu je určen nástroj scponly.

Po instalaci stejnojmenného balíčku stačí jediný jednoduchý krok - vyměnit výchozí shell uživatele zascponly, takto:

```
usermod -s /usr/bin/scponly uzivatel
```

Z bezpečnostních důvodů je nutné, aby daný uživatel neměl právo zápisu do svého domovského adresáře (aby nemohl upravovat konfigurační soubory v .ssh a potenciálně si tak otevřít vrátka k shellu):

```
chown root:root /home/uzivatel
chmod 755 /home/uzivatel
```

V rámci tohoto nastavení přesto vyvstává minimálně jeden další potenciální problém - uživatel může v tomto případě stále cestovat v adresářové struktuře mimo svůj domovský adresář. Pokud chcete zabránit i tomuto, je třeba využít chroot variantu scponly. Za tímto účelem je nejprve potřeba tuto verzi povolit:

```
dpkg-reconfigure scponly
```

Zde musím upozornit na to, že v takovém případě scponly získá setuid bit a bude spouštěn s právy uživatele root. Případná zranitelnost v scponly pak může mít katastrofální dopad (útočník získá rootovská oprávnění). Dalším krokem je užití pomocného skriptu v adresáři /usr/share/doc/scponly/setup_chroot s názvem setup_chroot.sh. Tento skript je před použitím třeba rozbalit:

```
cd /usr/share/doc/scponly/setup_chroot
gzip -d setup_chroot.sh.gz
chmod u+x setup_chroot.sh
./setup_chroot.sh
```

Skript za vás vytvoří uživatele, zkopíruje všechny nutné nástroje a knihovny do chrootu, zajistí úpravu práv domovského adresáře uživatele a vytvoří adresář, kam bude mít daný uživatel právo k zápisu. Naneštěstí, v Debianu Lenny tento skript neprovede jednu velmi důležitou operaci, a sice vytvoření /dev/null v chrootu, což je třeba provést ručně:

```
mkdir /home/uzivatel/dev
cp -a /dev/null /home/uzivatel/dev/
```

## Správa linuxového serveru: Linuxový firewall, základy iptables

Firewall v Linuxu je tvořen projektem Netfilter, který pracuje na úrovni jádra a umožňuje filtrovat pakety na základě mnoha kritérií. Základním nástrojem pro nastavení paketového filtru je známý řádkový nástroj iptables. Předesílám, že existuje řada projektů, která se snaží práci s iptables usnadnit, popřípadě generovat vlastní pravidla a uživatele od tohoto nástroje co nejvíce odstínit. Některé z těchto nástrojů časem představím, nicméně v tomto článku se budu zabývat samotným nástrojem iptables a principy, na jakých jeho funkce stojí.

Je nutné rovněž předeslat, že Netfilter není pouze firewall, je to především paketový filtr, který umožňuje provádět řadu věcí, z nichž jedna možnost je vytvořit pravidla, která budou zastávat funkci firewallu. Některé z funkcí, které nejsou typické pro firewall, nýbrž pro paketový filtr jako takový, vám v průběhu tohoto seriálu také představím, i když se budu zaměřovat primárně na užití Netfiltru pro konstrukci firewallu.

### Varování

Pokud se budete učit pracovat s iptables a experimentovat, určitě tak čiňte na počítači, ke kterému máte fyzický přístup. Pracovat s iptables na ostrém serveru v datacentru je velké riziko, pokud nevíte přesně, co děláte a jaké to bude mít důsledky. Může se velmi jednoduše stát, že si pod sebou "uříznete větev" a na server už se vzdáleně nepřihlásíte, abyste to mohli opravit.

### Základy fungování Netfiltru

#### *Řetězy a politika*

Jedna z dřívějších verzí nástroje iptables se jmenovala ipchains, a jsou to právě řetězy (chains), které tvoří pilíř fungování paketového filtru v Linuxu. Jejich funkci ilustruje následující obrázek:

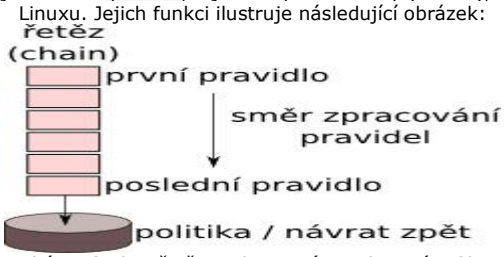



*Schéma funkce řetězu v linuxovém paketovém filtru*

Jak je patrné z obrázku, tyto řetězy jsou tvořeny jednotlivými pravidly. Paket, který je zachycen v některém z řetězů, putuje od prvního pravidla k následujícímu, dokud některému z pravidel nevyhoví. Pokud některému z pravidel vyhoví, provede se některá z možných akcí, v rámci které může být paket zahozen (DROP), odmítnut (REJECT), přijat (ACCEPT) nebo předán jinému řetězu.

Pokud nevyhoví žádnému z pravidel v daném řetězu, může se stát jedna ze dvou věcí. Jedná-li se o některý ze základních/vestavěných řetězů (viz dále), pak o jeho dalším osudu rozhodne politika daného řetězu (k dispozici jsou dvě možnosti: přijmout paket - ACCEPT nebo zahodit paket - DROP). Pokud se jedná o uživatelsky definovaný řetěz, je paket vrácen tam, odkud byl uživatelsky definovanému řetězu poslán (RETURN).

#### *Základní řetězy*

Základní vestavěné řetězy, primární řetězy, které tvoří základ fungování Netfiltru. Mezi tyto řetězy patří řetěz INPUT, řetěz FORWARD a řetěz OUTPUT. Jejich význam vám osvětlí následující obrázek:

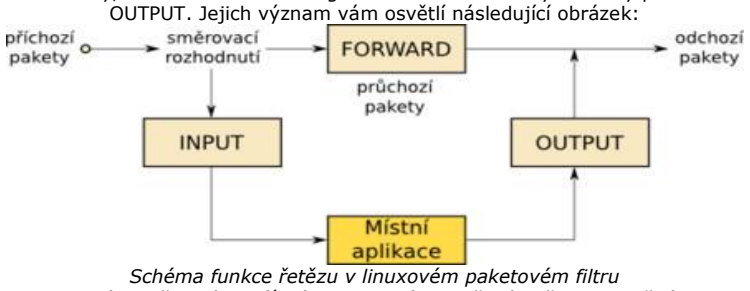



*Schéma funkce řetězu v linuxovém paketovém filtru*

Jak je z obrázku patrné, jakmile paket dorazí na některé ze síťových rozhraní a je předán přes ovladač jádru, je provedeno směrovací rozhodnutí. Je-li paket určen místnímu počítači, směřuje do řetězu INPUT. Je-li určen nějakému jinému počítači, směřuje do řetězu FORWARD. Dlužno dodat, že předávání paketů (tj. funkcionalita směrovače/routeru) je třeba explicitně povolit v jádře. Bez tohoto povolení pakety předávány nebudou, a průchozí pakety budou zahozeny. K tomu se ale ještě dostaneme. V tuto chvíli je pro nás podstatný řetěz INPUT, kam směřují pakety určené pro tento počítač. Pokud tyto pakety projdou řetězem INPUT a budou povoleny (ať již explicitně pravidlem nebo politikou), budou předány příslušné aplikaci.

Pro úplnost dodám, že existují kromě výše uvedených ještě dva další řetězy, PREROUTING a POSTROUTING, které jsem zamlčel, ale těm se budu blíže věnovat později.

#### Pravidla

V jednotlivých řetězech lze definovat pravidla, která provádí vlastní filtrování paketů. Tato pravidla mají určité podmínky, resp. kritéria, kterým musí paket vyhovět, a příslušnou akci, která se má provést, pokud paket pravidlu vyhověl. Kupříkladu:

```
iptables -A INPUT -s 1.2.3.4 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
```

Tyto dva příkazy přidají dvě pravidla na konec řetězu INPUT. První pravidlo zachytí pakety se zdrojovou adresou 1.2.3.4 (-s 1.2.3.4) a propustí je (-j ACCEPT), tj. příslušný paket již nebude procházet žádnými dalšími pravidly. Pokud paket toto kritérium nesplní, přijde na řadu druhé pravidlo, kterému vyhoví jakýkoliv TCP paket s cílovým portem 22 (-p tcp --dport 22), přičemž tyto pakety budou zahozeny (-j DROP). Pokud byste chtěli paket odmítnout, tj. sice zahodit, ale současně poslat zpět ICMP zprávu o důvodu odmítnutí, použili byste následující příkaz:

```
iptables -A INPUT -p tcp --dport 22 -j REJECT --reject-with icmp-admin-prohibited
```

A teď, kontrolní otázka. Podívejte se na následující sadu pravidel a popřemýšlejte, co se stane s TCP paketem směřujícím na port 22 - bude vygenerována ICMP zpráva o důvodu odmítnutí, nebo bude paket "tiše" zahozen?

```
iptables -A INPUT -s 1.2.3.4 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
iptables -A INPUT -p tcp --dport 22 -j REJECT --reject-with icmp-admin-prohibited
```

Správnou odpovědí je ona druhá možnost - takový paket bude zahozen, protože vyhoví druhému pravidlu z tohoto seznamu, bude zahozen a ke třetímu pravidlu, které by příslušnou ICMP zprávu vygenerovalo, se už nedostane.

#### *Přehled základních operací*

Nyní, když rozumíte základním principům práce s Netfiltrem, uvedu několik "taháků" se základními prvky pravidel. Nejprve přidávání a mazání pravidel:

```
# přidat pravidlo na konec řetězu INPUT
iptables -A INPUT ...
# přidat pravidlo na začátek řetězu INPUT
iptables -I INPUT ...
# odebrat pravidlo - specifikovat můžete pořadové číslo nebo pravidlo opsat:
iptables -D INPUT <pořadové číslo pravidla>
iptables -D INPUT ...
# vymazat všechna pravidla z řetězce INPUT
iptables -F INPUT
```

Odebrání pravidel by asi zasluhovalo bližší vysvětlení. Existující pravidlo může být vymazáno buď zadáním pořadového čísla pravidla, nebo prostě tak, že celé pravidlo opíšete, ale volbu -A nebo -I zaměníte za -D, takto:

```
# přidání pravidla
iptables -A INPUT -i eth0 -s 10.0.1.5 -p tcp --dport 22 -j ACCEPT
# odebrání pravidla
iptables -D INPUT -i eth0 -s 10.0.1.5 -p tcp --dport 22 -j ACCEPT
```

Pravidla si můžete nechat vypsat následujícím příkazem:

```
iptables -L -n -v
```

Volba `-L` zařídí vypsání pravidel, volba `-n` bude adresy a porty uvádět číselně a volba `-v` zařídí podrobnější výpis. Následuje seznam nejčastěji používaných kritérií:

```
# zdrojová adresa
-s 1.2.3.4
# cílová adresa
-d 4.3.2.1
# TCP paket, cílový port 25
-p tcp --dport 25
# UDP paket, zdrojový port 53
-p udp --sport 53
# paket přichází z rozhraní eth0
-i eth0
# paket směřuje na rozhraní eth1
-o eth1
```

A na úplný závěr výpis základních akcí, které lze s pakety provádět.

```
# přijmout paket
-j ACCEPT
# odmítnout paket s příslušnou ICMP zprávou
-j REJECT --reject-with <typ ICMP zprávy>
# tiše zahodit paket
-j DROP
```

***Příklad nejjednoduššího, nestavového firewallu***

Tento díl zakončím příkladem nejjednoduššího firewallu, který lze sestavit. Nejprve vyčistím všechna pravidla:

```
iptables -F
```

Nyní specifikuji jednotlivá pravidla. Povolím veškerý provoz z místního rozsahu 10.0.0.0/8. Přístup k SSH povolím pouze z IP adresy 11.22.33.44. Dále povolím přístup k webserveru a k SMTP serveru.

```
iptables -A INPUT -s 10.0.0.0/8 -j ACCEPT
iptables -A INPUT -s 11.22.33.44 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 25 -j ACCEPT
```

Nakonec specifikuji výchozí politiku ve všech třech základních řetězech. Pro konstrukci firewallu se hodí strategie "vše zakázat, a povolit to nejnutnější", takže nastavím u řetězu INPUT politiku DROP:

```
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
```

Tímto bych tento díl ukončil. Příště vám představím stavový firewall a ukážu, jak pomocí něj zkonstruovat podstatně dokonalejší formu firewallu než tu, kterou jsem představil výše.

## Správa linuxového serveru: Linuxový firewall, základy iptables II

V tomto dílu se budu věnovat stavovému filtrování paketů a vytvoření velmi jednoduchého, v praxi použitelného firewallu pomocí Netfiltru. Upozorním také na to, jaká existuje podpora Netfiltru pro IPv6 a jak firewall nastavit pro IPv4 i IPv6.

### Stavový firewall

Jedna ze stěžejních výhod Netfilteru je schopnost rozlišovat, ke kterým existujícím spojením paket patří, pokud k nějakým takovým vůbec patří, a filtrovat jej na základě této příslušnosti. Pomocí této vlastnosti je možné pravidla pro filtrování paketů vztáhnout pouze na žádosti o vytvoření nového spojení, zatímco všechny pakety náležející již vytvořeným spojením rovnou propustit.Stavové filtrování zajišťuje modul state, který rozlišuje čtyři stavy:

- NEW - paket vytváří nové spojení nebo se vztahuje ke spojení, kde dosud neproběhla obousměrná komunikace
- ESTABLISHED - paket se vztahuje ke spojení, kde probíhá obousměrná komunikace (tedy již k vytvořenému spojení)
- RELATED - paket vytváří nové spojení, ale vztahuje se k některému z existujících (např. u FTP)
- INVALID - paket se nevztahuje k žádnému známému spojení (obvykle je to paket, který tu nemá co dělat, a proto je dobré ho rovnou zahazovat)

Samotný firewall využívající stavového filtrování může vypadat třeba takto:

```
1.  iptables -P INPUT DROP
2.  iptables -P OUTPUT ACCEPT
3.  iptables -P FORWARD DROP
4.  iptables -A INPUT -i lo -j ACCEPT
5.  iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
6.  iptables -A INPUT -m state --state INVALID -j DROP
7.  iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
8.  iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
9   iptables -A INPUT -p icmp -j ACCEPT
10. iptables -A INPUT -j REJECT --reject-with icmp-admin-prohibited
```

První tři řádky specifikují politiku (viz minulý díl), čtvrtý řádek povoluje všechna příchozí spojení z místní smyčky (local loopback). Toto pravidlo je nesmírně důležité, neboť bez možnosti síťové komunikace po místní smyčce nebude ledacos správně fungovat, včetně třeba X serveru). Je tedy třeba explicitně povolit provoz místní smyčky všude tam, kde je politika pro řetěz INPUT či OUTPUT nastavena na DROP. Jen pro úplnost, pro řetěz OUTPUT by příslušné pravidlo vypadalo takto:

```
iptables -A OUTPUT -o lo -j ACCEPT
```

Pátý řádek využívá modulu state a propouští všechny pakety, které náleží již vytvořeným spojením (ESTABLISHED) nebo novým spojením, která k nim patří (RELATED).Šestý řádek zahazuje "neplatné" (INVALID) pakety, tedy pakety, které nepatří k žádným spojením. Zahazovat takové pakety je obvykle v pořádku, jen je třeba zmínit, že sem mohou patřit i pakety, o kterých systém pro analýzu stavů (connection tracking aneb conntrack) ztratí přehled. To se může stát třeba v případě, že dané spojení bude neaktivní (neproteče paket v žádném směru) déle, než je nastaven příslušný timeout. Pak conntrack dané spojení z tabulky vyřadí, a pokud se poté objeví paket náležející k danému zahozenému spojení, bude považován právě za INVALID.Jen tak pro zajímavost, zobrazením obsahu tohoto souboru je možné nechat si vypsat všechna spojení, o kterých má conntrack přehled:

```
/proc/net/ip_conntrack
```

Timeoutů v rámci conntracku je více, pro vytvořená (ESTABLISHED) TCP spojení je příslušná hodnota v souboru:

```
/proc/sys/net/netfilter/nf_conntrack_tcp_timeout_established
```

V tomto adresáři naleznete mnohá nastavení conntracku, včetně maximálního počtu spojení, o kterých siconntrack vede záznam (nf_conntrack_max). Za tuto menší odbočku se omlouvám, ale při ladění některých nepříjemných problémů se stavovým firewallem se vám může tato informace hodit. Zejména na domácích routerech, které obsahují Linux, může být záhodno některé z těchto hodnot prověřit, zejména, pokud vám "zamrzají" spojení - někdy totiž bývají nastavena příliš nízko (kvůli šetření paměti).

Ale teď už zpět k příkladu výše. Řádky 7 a 8 povolují nová TCP spojení směřující na porty 22 (SSH) a 80 (HTTP). Pokud by se v tomto případě jednalo o webserver, asi by bylo vhodné analogickým způsobem povolit další porty jako třeba HTTPS či SMTP. Devátá řádka povoluje příchozí ICMP pakety. I když to není úplná nutnost (resp. server bude fungovat a bude přístupný i bez toho), hodí se povolit minimálně ping, tedy ICMP echo request. Je samozřejmě možné specifikovat pouze tento typ ICMP paketů a ostatní již nebrat v úvahu:

```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
```

Poslední pravidlo v příkladu výše zařídí, že paket, který nevyhoví žádným předešlým pravidlům, bude odmítnut, tedy sice zahozen, ale odesílateli se vrátí příslušná ICMP zpráva o "nedoručení". Pokud by tam toto pravidlo nebylo, pak by bylo s paketem nevyhovujícím žádnému z pravidel naloženo dle politiky daného základního řetězu, tedy v tomto případě by byl zahozen (politika řetězu INPUT je nastavena na DROP, viz první řádek příkladu). Prosté zahazování paketů působí problémy klientům, kteří musejí při pokusu připojit se na filtrovaný port čekat, než vyprší příslušný timeout. Toto poslední pravidlo sice případným útočníkům o něco malinko usnadňuje práci, ale útočník si porty oskenuje, i když budou pakety směřující na nepovolené porty zahazovány - jen mu to bude trvat o něco déle.

### IPv6 a Netfilter

V souvislosti s rychle docházejícím adresním prostorem IPv4 a neodvratným přechodem na IPv6 možná stojí za to uvést, že Netfilter je pro IPv6 již delší dobu plně připraven, ovšem s tím důležitým faktem, že filtry pro IPv4 a IPv6 provoz jsou oddělené. Proto, pokud jste nastavili firewall prostřednictvím nástroje iptablesa divíte se, proč se přes IPv6 dostanete bez problémů i k filtrovaným službám, je to právě z tohoto důvodu - filtr pro IPv6 jste totiž ještě nenastavili. Ten se nastavuje pomocí nástroje ip6tables, jehož syntaxe je prakticky totožná s nástrojem iptables.

U většiny pravidel je možné ponechat úplně stejně argumenty, pouze někde je třeba zohlednit rozdíly mezi protokoly IPv4 a IPv6. Jedním z takových míst je úloha protokolu ICMP - v rámci IPv6 totiž na úrovni ICMP probíhá třeba autokonfigurace (neighbour discovery, atd.), takže pokud nemáte ve firewallu povolený ICMP jako takový (nebo alespoň jeho příslušné typy), můžete se setkat s tím, že po nastavení IPv6 firewallu se zakázaným ICMP vám IPv6 konektivita vypadne. Protokol ICMP v rámci IPv6 můžete povolit takto:

```
ip6tables -A INPUT -p ipv6-icmp -j ACCEPT
```

Firewall uvedený výše by ve své IPv6 variantě vypadal po menší úpravě takto (rozdíly v argumentech jsou pouze na pátém a desátém řádku):

```
1.  ip6tables -P INPUT DROP
2.  ip6tables -P OUTPUT ACCEPT
3.  ip6tables -P FORWARD DROP
4.  ip6tables -A INPUT -i lo -j ACCEPT
5.  ip6tables -A INPUT -p ipv6-icmp -j ACCEPT
6.  ip6tables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
7.  ip6tables -A INPUT -m state --state INVALID -j DROP
8.  ip6tables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
9.  ip6tables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
10. ip6tables -A INPUT -j REJECT --reject-with icmp6-adm-prohibited
```

### Jak integrovat firewall do systému

Na rozdíl od mnoha jiných distribucí nemá Debian ve startovacích skriptech zohledněno nastavování firewallu, takže vám nezbyde než si případný firewall napsat jako shellový skript a začlenit ho do bootovacího procesu. Možností, jak toto provést je mnoho, včetně možnosti napsat si vlastní init skript a startovat firewall jako kteroukoliv jinou systémovou službu. Já zde předvedu tu úplně nejjednodušší - příslušný skript lze umístit jako pre-up skript v konfiguračním souboru pro síťová rozhraní, kterým je/etc/network/interfaces, takto:

```
iface eth0 inet static
address 10.0.0.254
netmask 255.255.255.0
gateway 10.0.0.138
pre-up /root/firewall.sh
```

## Správa linuxového serveru: Linuxový firewall, základy iptables III

V tomto dílu proberu použití vlastních řetězců ke zpřehlednění firewallu, možnosti logování paketů a filtrování předávaných (routovaných) paketů.

### *Vlastní řetězce*

Netfilter umožňuje definovat vlastní řetězce, a tím pravidla paketového filtru logicky členit a zjednodušit. Je to podobné jako práce s funkcemi v programovacím jazyce. Jak už víte, Netfilter zná tři základní řetězce:INPUT, OUTPUT a FORWARD. Tyto řetězce nelze smazat, pouze vyčistit. Je možné vytvořit nový řetězec a v existujících řetězcích se na něj pak odkázat, tedy poslat paket, který vyhoví danému pravidlu, do daného řetězce. Uživatelem definované řetězce nemají politiku, takže pokud paket uživatelským řetězcem projde, aniž by byl zachycen nějakým pravidlem, vrací se zpět do řetězce, odkud byl do uživatelsky definovaného řetězce odeslán.

Asi nejlepší bude demonstrovat tuto funkci na příkladu. Za tímto účelem rozšířím firewall, který jsem vám předvedl v minulém díle:

```
1.  iptables -F
2.  iptables -X
3.  iptables -P INPUT DROP
4.  iptables -P OUTPUT ACCEPT
5.  iptables -P FORWARD DROP

6.  iptables -N ssh
7.  iptables -A ssh -s 10.0.0.0/8 -j ACCEPT
8.  iptables -A ssh -s 43.21.12.34 -j DROP
9.  iptables -A ssh -m limit --limit 5/sec --limit-burst 100 -j ACCEPT
10. iptables -A ssh -j DROP

11. iptables -A INPUT -i lo -j ACCEPT
12. iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
13. iptables -A INPUT -m state --state INVALID -j DROP
14. iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ssh
15. iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
16. iptables -A INPUT -p icmp -j ACCEPT
17. iptables -A INPUT -j DROP
```

Příkaz na prvním řádku odstraní všechna pravidla ze všech řetězců, druhý příkaz odstraní všechny prázdné uživatelem definované řetězce. Následuje vytvoření nového řetězce s názvem ssh, kde je naznačen whitelist a blacklist (řádky 7 a 8), a poté následuje omezení počtu spojení na SSH (řádek 9). Poslední pravidlo v řetězci (řádek 10) zahazuje všechny pakety, které neodpoví některému z pravidel v řetězci.

Na řetězec ssh je odkázáno na řádku 14, kde je uživatelsky definovaný řetězec předán parametru -j, který určuje, co se má s paketem vyhovujícím danému pravidlu provést. Příchozí pakety tedy v případě takto definovaného firewallu dorazí nejprve do řetězce INPUT, tedy na řádek č. 11, a teprve pakety, které vyhoví pravidlu na řádku č. 14, budou poslány do řetězce ssh.

Na další příklad užití vlastních řetězců (a o něco lepší metodu zabezpečení SSH na úrovni firewallu) se můžete podívat ve čtvrtém dílu miniseriálu o zabezpečení SSH.

Pokud potřebujete paket z řetězu propustit, aniž by byl přijat, odmítnut nebo zahozen, tedy aby se vrátil zpět do původního řetězu, odkud byl do daného uživatelem definovaného řetězu vyslán, použijte RETURN jako parametr pro -j, takto:

```
[...]

1.  iptables -N whitelist
2.  iptables -A whitelist -s 10.0.0.0/8 -j RETURN
3.  iptables -A whitelist -j DROP

[...]

4.  iptables -A INPUT [...] -j whitelist

[...]
```

V tomto případě bude paket vyhovující čtvrtému pravidlu poslán do řetězce whitelist, kde projde pravidlem č. 2, přičemž pokud mu vyhoví, poputuje zpět do řetězce INPUT za pravidlo č. 4, pokud mu nevyhoví, pak dorazí k pravidlu č. 3 a bude zahozen.

### Logování paketů

Někdy může být velice žádoucí, abyste určitý specifický síťový provoz monitorovali trošku blíže, tedy abyste nechali jisté pakety logovat. Pokud se ještě chvilku budu držet příkladů na vlastní řetězce, mohl by příklad na logování paketů vypadat třeba takto:

```
[...]

1.  iptables -N
2.  iptables -A strange -s 10.0.0.0/8 -j LOG --log-prefix "iptables-strange-local: "
3.  iptables -A strange -s 12.23.34.45 -j LOG --log-prefix "iptables-strange-offender: "
4.  iptables -A strange -j DROP

[...]

5   iptables -A INPUT -m state --state INVALID -j strange

[...]
```

V tomto příkladu jsou všechny pakety ve stavu INVALID poslány do uživatelsky definovaného řetězcestrange, ve kterém dochází k výběru jedné sítě a jedné IP adresy, u nichž se provádí logování. Všechno ostatní se zahazuje.

Samotné logování je průchozí operace, tzn. pokud paket pravidlu s cílem LOG vyhoví, je zalogován, ale pokračuje dál. Proto je zde na řádku č. 4 definováno pravidlo, které všechny pakety v tomto řetězci zahodí. Samotné logování má nepovinnou volbu --log-prefix, která umožňuje specifikovat, čím bude hlášení na řádce se záznamem v logu začínat. Výsledek pak v logu může vypadat třeba takto:

```
Jul 25 03:49:51 debian kernel: [5466089.604926] iptables-strange-local: IN=eth0 OUT=eth1 SRC=10.0.1.15 DST=1.2.3.4 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=22177 DF PROTO=TCP SPT=45320 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0 UID=0 GID=0
```

Množství informací, které jsou o daných paketech získávány, můžete ovlivnit volbou --log-level.

### *Malý příklad na procvičení*

Podívejte se na následující příklad a zauvažujte, co se stane s paketem, který je ve stavu INVALID a má zdrojovou IP adresu 10.0.1.12?

```
1.  iptables -P INPUT ACCEPT

2.  iptables -N
3.  iptables -A strange -s 10.0.0.0/8 -j LOG --log-prefix "iptables-strange-local "
4.  iptables -A strange -s 12.23.34.45 -j LOG --log-prefix "iptables-strange-offender "

5.  iptables -A INPUT -i lo -j ACCEPT
6.  iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
7.  iptables -A INPUT -m state --state INVALID -j strange
```

Paket dorazí do řetězce INPUT, kde je zachycen pravidlem na řádku č. 7 a odeslán do řetězce strange. Zde vyhoví pravidlu na řádku č. 3 a je zalogován. Jelikož logování je průchozí operace, pokračuje paket po zalogování dál, avšak v řetězci strange již nejsou žádná další pravidla, tudíž

bude příslušný paket vrácen zpět do řetězce INPUT. Zde již také nejsou žádná pravidla, tudíž bude s paketem naloženo dle výchozí politiky pro řetězec INPUT, a paket bude propuštěn, což je samozřejmě v případě INVALID paketů patrně nežádoucí.

***Řetězec FORWARD a routing v GNU/Linuxu***

GNU/Linux může fungovat jako router, tedy předávat pakety z jednoho rozhraní na jiné, z jedné sítě do druhé. Pakety, které nejsou určeny pro daný počítač, nýbrž jím jen prochází, putují v rámci paketového filtru řetězcem FORWARD. To je důvodem, proč v dřívějších příkladech byla politika pro řetězec FORWARD nastavena na DROP.

K samotnému zprovoznění routingu je však zapotřebí routing povolit v konfiguraci jádra, tedy v sysctl.conf:

```
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
```

Všimněte si, že je zde opět oddělen protokol IPv4 a IPv6, tzn. předávání paketů se nastavuje individuálně pro každý z protokolů. Změna tohoto konfiguračního souboru se projeví až po restartu, i když je samozřejmě možné příslušnou změnu provést ručně a okamžitě, prostřednictvím nástroje sysctl:

```
sysctl -w net.ipv4.ip_forward=1
sysctl -w net.ipv6.conf.all.forwarding=1
```

Samotné filtrování předávaných paketů pak může vypadat třeba takto:

```
[...]

1. iptables -P FORWARD DROP
2. iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
3. iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
4. iptables -A FORWARD -s 10.0.0.0/24 -d 10.0.1.0/24 -j ACCEPT
5. iptables -A FORWARD -s 10.0.1.0/24 -d 10.0.0.0/24 -j ACCEPT

[...]
```

Zde je na prvním řádku nastavena výchozí politika řetězce FORWARD, na druhém řádku je povolen průchod paketů, které náleží již vytvořeným spojením, nebo se jich týkají, na třetím řádku je pak povolen veškerý síťový provoz směřující z rozhraní eth0 na rozhraní eth1, a konečně na posledních dvou řádcích je povolen provoz v obou směrech mezi sítěmi 10.0.0.0/24 a 10.0.1.0/24.

***Alternativy k iptables***

Nastavovat firewall je komplikovaná záležitost, což je patrně z dosavadních dílů o základech iptables více než jasné. Proto existuje řada projektů, které si kladou za cíl stavbu firewallu zjednodušit. Existují dokonce i jisté ekvivalenty "osobních" firewallů, které se snaží být uživatelsky přívětivé a zaměřují se na běžné uživatele. Tím je třeba nástroj Firestarter.

Jiné typy nástrojů se pokouší usnadnit vytváření pravidel, ať již snahou o usnadnění návrhu komplikovaného firewallu (fwbuilder), nebo značným zjednodušením syntaxe (ufw, FireHOL, atd.).

Na závěr jsem si nechal projekt Shorewall, což je de facto nadstavba nad iptables, která z několika přehledných konfiguračních souborů vygeneruje příslušná pravidla pro nástroj iptables a nastaví linuxový paketový filtr. Pokud se chcete o Shorewallu dozvědět více, můžete se podívat na článek Shorewall 2 - rychlá obrana na stanici a dvoudílný miniseriál o Shorewallu

## Správa linuxového serveru: Linuxový firewall, základy iptables IV

V tomto díle se dozvíte o tabulkách v Netfiltru, o možnostech práce s NATem, o přesměrování portů či maškarádě a o některých dalších zajímavých modulech Netfiltru, které vám pomohou při stavbě pokročilejších firewallů.

### Tabulky v Netfilteru

Už z názvu nástroje iptables lze odvodit, že paketový filtr v Linuxu pracuje s tabulkami. Tuto informaci jsem vám dosud tajil, abych vám pochopení základů příliš nekomplikoval. Nyní je ale čas, abych se podíval na jednotlivé tabulky v linuxovém paketovém filtru, a řekl vám, k čemu jsou. Pokud v rámci použití nástroje iptables nezadáte název tabulky, se kterou chcete pracovat, využije se výchozí tabulka filter, v rámci které probíhá samotné filtrování paketů. Kromě tabulky filter existují ještě tři další - tabulka nat, tabulka mangle a tabulka raw. V každé z těchto tabulek jsou různé výchozí řetězy, se kterými můžete pracovat. Tabulka mangle slouží k modifikování nebo značkování paketů (užívá se např. v rámci QoS). Tabulka nat se používá pro překlad adres a tabulka raw se používá pro označení paketů, které se mají vyhnout sledování spojení (connection tracking). Jelikož tento miniseriál je spíše orientován na základy linuxového paketového filtru, nebudu probírat všechny tabulky se všemi řetězy, které se v nich nachází. Místo toho se zaměřím na tabulku nat a dva řetězy v ní, které nám trošku rozšíří schéma procházení paketů Netfiltrem. Pokud se chcete o jednotlivých tabulkách dozvědět více, podívejte se na odkazy pod článkem.


*Jak procházejí pakety linuxovým paketovým filtrem*

Na tomto obrázku jsou vidět dva další výchozí řetězce, které patří tabulce nat (vyznačené zeleně). Jsou jimi řetězy PREROUTING a POSTROUTING. Řetěz PREROUTING v tabulce nat zachytává pakety ještě před směrovacím rozhodnutím, a řetěz POSTROUTING zachytává pakety těsně před tím, než opustí daný počítač. Tyto dva řetězy využijete, pokud si chcete postavit nějakou formu NATu (viz níže).

Raději ještě doplním varování, abyste filtrování paketů neprováděli v tabulce NAT místo v řetězechINPUT a OUTPUT v tabulce filter, jelikož některé pakety tabulku nat "obchází".

### Netfiltr a NAT

NAT znamená "Network address translation", tedy překlad adres. Existují dvě formy NATu, zdrojový NAT (SNAT), kde se u paketů mění zdrojová adresa, a cílový NAT (DNAT), v rámci kterého dochází k úpravě cílové adresy. Speciálním případem SNATu je maškaráda, která je vhodná pro připojení s dynamickou IP adresou.Je jasné, že jednotlivé formy NATu se uplatní pouze v jednom z řetězců tabulky nat. Cílový NAT, tedy DNAT, uplatníte pouze v řetězci PREROUTING, zatímco zdrojový NAT v řetězci POSTROUTING. V těchto řetězcích pak můžete pro zpřesnění využít specifikaci vstupních nebo výstupních síťových rozhraní (parametry -o a -i). S tím ale pozor, v řetězci PREROUTING není možné se odkazovat na výstupní rozhraní (parametr -o), neboť ještě nebylo rozhodnuto, kam bude paket směřován. Stejně tak vstupní rozhraní (parametr -i) nelze použít v řetězci POSTROUTING.

Překlad adres využijete asi nejčastěji ve dvou podobách - přesměrování portů a maškaráda (za kterou "schováte" třeba domácí síť před svým poskytovatelem).

#### *Přesměrování portů a DNAT*

Přesměrování portů využijete v mnoha oblastech, ať už v případě domácího serveru, kde dostanete na routeru jednu veřejnou IP adresu, a budete chtít přesměrovat některé porty na server v domácí síti (nejlépe v demilitarizované zóně), nebo v případě, že máte na jednom počítači více virtuálních strojů, a chcete, aby některé jejich služby byly přístupné na IP adrese daného fyzického počítače. Ideální by samozřejmě bylo získat více IP adres a adresovat každý z počítačů ve vnitřní/virtuální síti a řídit přístup k nim prostřednictvím firewallu, ale toto není bohužel všude možné. Podívejte se na následující příklad přesměrování portů:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j DNAT --to 10.0.2.200:8080
iptables -A FORWARD -i eth1 -o eth2 -d 10.0.2.200 -p tcp --dport 8080 -j ACCEPT
```

Na prvním řádku je specifikováno pravidlo pro přesměrování portu. Toto pravidlo změní cílovou adresu a cílový port paketů směřujících na port 80 rozhraní eth1 na IP adresu 10.0.2.200 a port 8080. Všimněte si specifikace tabulky nat (-t nat). Výše zmíněné pravidlo změní pouze cílovou adresu paketu, který pravidlu vyhovuje, nic jiného s paketem nedělá. Takový paket pak projde směrovacím rozhodnutím a putuje do řetězce FORWARD, kde je třeba zajistit jeho průchod (viz druhé pravidlo, které umožní takovému paketu odcestovat rozhraním eth2). Nezapomeňte také na to, že je třeba povolit směrování paketů jako takové, viz minulý díl seriálu. V rámci DNATu samozřejmě není nutné specifikovat cílový port, takže je možné přesměrovat veškerou komunikaci:

```
iptables -t nat -A PREROUTING -i eth1 -d 10.0.0.1 -j DNAT --to 10.0.2.200
```

Toto pravidlo přesměruje veškerý síťový provoz proudící na rozhraní eth1 a určený pro IP adresu 10.0.0.1na IP adresu 10.0.2.200.

#### *Maškaráda a SNAT*

Pokud jste v situaci, kdy vám poskytovatel přidělí dynamicky IP adresu třeba na rozhraní modemu ppp0, a vy chcete za tento jeden počítač "schovat" celou vnitřní síť, použijete maškarádu. Ta zajistí, že se u všech odchozích paketů směřujících na dané rozhraní změní zdrojová IP adresa na IP adresu přidělenou danému rozhraní:

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Maškaráda je podoblastí zdrojového NATu a oproti klasickému cíli SNAT v rámci iptables má dva zásadní rozdíly. Jedním je absence nutnosti specifikovat zdrojovou adresu (použije se IP adresa výstupního rozhraní), druhým je pak skutečnost, že při shození daného rozhraní Netfilter "zapomene" informace o všech relevantních spojeních. Pokud se adresa daného rozhraní nemění, pak místo maškarády použijte klasický SNAT:

```
iptables -t nat -A POSTROUTING -o eth2 -j SNAT --to 10.0.5.15
```

Toto pravidlo změní zdrojovou IP adresu všech odchozích paketů směřujících na rozhraní eth2 na10.0.5.15.

### Některé další moduly Netfiltru

Netfilter nabízí celou řadu modulů, které umožňují filtrovat pakety dle řady doplňujících kritérií. Na závěr miniseriálu o Netfiltru ještě představím tři z nich.

#### *Vlastník paketu (owner)*

Na serveru nemusí být úplně od věci monitorovat nejenom příchozí komunikaci, ale i odchozí. V rámci této odchozí komunikace se vám může hodit filtrovat pakety na základě UID nebo GID procesu, který je generuje:

```
iptables -A OUTPUT -o eth0 -m owner --uid-owner podezrely-uzivatel -m limit --limit 1/s --limit-burst 10 -j LOG
```

Toto pravidlo bude logovat odchozí pakety generované procesy uživatele podezrely-uzivatel a směřující na síťové rozhraní eth0. Naznačil jsem zde i ochranu proti přeplnění logů, pokud by uživatel generoval příliš velký síťový traffic. Ta je zajištěna modulem limit. Pokud byste chtěli logovat provoz všech uživatelů (členů skupiny users), specifikovali byste příslušnou skupinu takto:

```
iptables -A OUTPUT -o eth0 -m owner --gid-owner users -m limit --limit 1/s --limit-burst 10 -j LOG
```

#### *Počet spojení (connlimit)*

V některých situacích se vám může hodit zaškrtit určitý provoz, pokud překoná určitý počet spojení:

```
iptables -A FORWARD -i eth2 -o eth0 -m connlimit --connlimit-above 10 -j REJECT
```

Toto pravidlo, použité na směrovači, omezí síťový provoz z rozhraní eth2 na eth0 tak, že povolí pouze 10 spojení. Ostatní pak bude odmítat.

#### *Multiport*

Pokud chcete v jednom pravidlu specifikovat více TCP/UDP portů, můžete použít modul multiport:

```
iptables -A INPUT -p tcp -m multiport --ports 22,80,40000:50000 -j ACCEPT
```

Zde je naznačeno jak specifikování několika portů za sebou (oddělovač je čárka), tak specifikování rozsahu (oddělovačem rozsahu je dvojtečka). Takto je možné specifikovat až 15 portů (rozsah je počítán jako dva porty).

#### *Další moduly*

Informace o modulech Netfiltru jsou dobře popsány v manuálových stránkách nástroje iptables, kam vám určitě doporučuji někdy zavítat.

## Správa linuxového serveru: Jak aktualizovat server(y)

Každý operační systém a software na něm je třeba aktualizovat. U serveru to platí dvojnásob, jelikož je permanentně vystaven útokům z celého světa. Přitom aktualizace serveru nebývají úplně triviální. V tomto dílu se dozvíte více o této problematice a doporučených postupech.

### Předmluva

Oproti aktualizacím desktopu je aktualizace serveru přeci jen v pár věcech odlišná. Na desktopu je obvykle ve výchozí instalaci přítomen nějaký nástroj, který vás na dostupnost nových aktualizací upozorní a provede vás příslušným procesem, a to obvykle v pohodlí grafického prostředí. Na serveru si upozorňování i provádění aktualizací musíte zařídit sami. Druhým podstatným rozdílem je dopad případného problému při aktualizaci - na serveru obvykle jakýkoliv problém postihuje řádově větší množství lidí.

Na základě vlastní zkušenosti si troufám odhadnout, že v případě stabilní větve Debianu je pravděpodobnost nějakého problému při aktualizaci velmi malá, správci se snaží o maximální stabilitu, takže aktualizace proběhnou v drtivé většině případů plně automaticky, bez nutnosti intervence správce. Samozřejmě tomu tak není úplně vždy, někdy je už z povahy aktualizace nutná intervence správce, a to i ve stabilní větvi distribuce, nehledě na různé testovací větve či rolling release distribuce, kde hrozí "rozbití" něčeho prakticky při každé aktualizaci. Proto není u serveru úplně vhodné provádět aktualizace automaticky třeba v rámci cronu, i když to samozřejmě technicky proveditelné je.

Doufám, že nemusím příliš zdůrazňovat, že opačný přístup, tedy neprovádění aktualizací vůbec, je téměř jistá sebevražda. Už jsem měl tu čest s jedním serverem, kde se na aktualizaci distribuce čekalo tak dlouho, až server úspěšně napadl útočník a převzal nad ním kontrolu.

### Druhy aktualizace a aktualizace v Debianu

Aktualizace lze členit dle více kritérií. Předně je třeba rozlišovat aktualizace v rámci určitého vydání distribuce (tj. třeba v rámci Debianu stable), aktualizace distribuce na novější verzi (tj. třeba aktualizaci Debianu 4.0 na Debian 5.0), a aktualizaci rolling release distribuce.

Dále je třeba rozlišovat povahu aktualizace, tedy jedná-li se o bezpečnostní aktualizaci, nebo pouze o opravu nějakého problému, který nemá bezpečnostní dopad, nebo jedná-li se o aktualizaci na novou verzi dané komponenty.

Aktualizace stabilní větve distribuce by měla být relativně bezproblémová, nemělo by docházet k tomu, aby aktualizace nějakého balíčku rozbila nějakým způsobem nakonfigurovanou službu. Do stabilní větve distribuce (alespoň u Debianu to tak je) by se měly dostávat pouze nutné opravy a bezpečnostní aktualizace, ne nové verze balíčků, kde by třeba vlivem změny syntaxe konfiguračních souborů došlo k narušení nakonfigurovaných služeb.

Aktualizace na nové vydání distribuce je naopak poměrně komplikovaná záležitost. Jelikož se mění verze balíčků a mezi dvěma stabilními vydáními (nejen Debianu) je obvykle velká časová propast, je dost dobře možné, že se v rámci aktualizace nevyhnete problémům, kdy něco přestane fungovat (bude potřeba upravit konfigurační soubor, nebo samotnou databázi uchovávající data atd.). Abyste možnost výskytu takového problému omezili na minimum, doporučuji s aktualizací vyčkat rozumně dlouhou dobu po vydání nové verze distribuce, než se pokusíte provést upgrade. Během této doby se snad většina problémů buď vyřeší, nebo se na ně alespoň přijde a vy se na ně budete moci připravit.

Před samotným upgradem se určitě podívejte po nějaké dokumentaci, a to jak po dokumentaci vztahující se k samotnému optimálnímu postupu aktualizace, tak po poznámkách k vydání, které bývají obzvláště důležité, neb ukrývají seznam známých problémů, na které můžete při aktualizaci narazit.

V případě produkčních systémů bývá vhodné aktualizaci provést pro jistotu "nanečisto" na podobném systému nebo na kopii produkčního systému postavené třeba ze záloh, výsledek důkladně zkontrolovat a otestovat, a teprve pak aktualizovat ostrý server. Samotnou aktualizaci bývá velmi vhodné naplánovat v čase, kdy je server minimálně vytížen, resp. minimálně využíván. Zálohování (nejen) těsně před aktualizací je v tomto případě, jak pevně doufám, samozřejmost.

Rolling release distribuce mají dost specifický charakter - do repositářů se dostávají rovnou nové verze programů (obvykle po otestování), což znamená, že každá aktualizace v sobě skrývá nebezpečí, že něco přestane fungovat. Z tohoto důvodu nebývají tyto typy distribucí příliš doporučovány na produkční servery, i když svědomitý a schopný správce jistě dokáže s jistou dávkou úsilí většině problémů úspěšně předejít. Jen to asi dá trošku více námahy.

#### *Aktualizace v Debianu*

V případě Debianu se aktualizace provádí následovně:

```
aptitude update && aptitude full-upgrade
```

První část příkazu aktualizuje databázi balíčků (a stav balíčků v repositářích), druhá část provede vlastní aktualizaci. Tento příkaz by měl pokrýt i situace, kdy se mění závislostní vazby a kdy vyvstane nutnost nějaký balíček odstranit, aby bylo možné zaktualizovat úplně všechno. Bezpečnou alternativou, v rámci které se aktualizace omezí pouze na případnou instalaci nového balíčku, ale nikoliv už na odstranění některého nainstalovaného, je safe-upgrade:

```
aptitude update && aptitude safe-upgrade
```

Samotný safe-upgrade má nepovinný parametr --no-new-installs, který zabrání i instalaci nových balíčků v rámci aktualizace.

Domnívám se, že největší nebezpečí spojené s full-upgrade spočívá spíše v situaci, kdy máte buď přidané některé neoficiální repositáře (kde se mohou tlouci balíčky s oficiálními repositáři), nebo když se pokoušíte aktualizovat na novější verzi distribuce. U aktualizací v rámci stabilní větve Debianu by ke konfliktům docházet nemělo.

Pokud dojde k nějakému konfliktu, a je potřeba nějaký balíček odstranit, nabídne vám Aptitude řešení. V každé takové situaci byste měli zbystřit a dávat pozor, s čím souhlasíte.

### Strategie pro aktualizaci

První z možností je automatická aktualizace, tj. příslušný záznam v cronu, který pravidelně aktualizuje systém. Z důvodů popsaných výše důrazně nedoporučuji něco takového používat na produkčních serverech. O něčem podobném bych uvažoval u domácího serveru či u osobního VPS, kde tolik nevadí, pokud třeba něco přestane fungovat. Kromě ručního vytvoření skriptu, který vám toto zajistí, můžete využít balíček cron-apt.

Druhou z možností je ruční aktualizace. Ta samozřejmě předpokládá nutnost dozvědět se o dostupnosti nových aktualizací. Za tímto účelem můžete použít výše zmíněný cron-apt nebo také apticron. Tyto nástroje vás budou informovat mailem o nových aktualizacích.

Ve spolupráci s balíčkem apt-listchanges umí apticron posílat v mailech i podrobnosti o aktualizacích. Dozvíte se tak třeba, zdali se jedná o bezpečnostní aktualizaci, čeho se aktualizace týká a také, jak moc je doporučováno aktualizovat.

V rámci ruční aktualizace máte samozřejmě možnost vybírat, kdy a co aktualizovat. Jedna z možných strategií je aktualizovat vše, druhou možností je aktualizovat jen to, co je opravdu nutné. K tomuto rozhodování vám velice dobře poslouží výše zmíněné nástroje. Nemusí být také úplně od věci přihlásit se k odběru zpráv o bezpečnostních problémech a aktualizacích. V případě Debianu se podívejte na

## Správa linuxového serveru: Pár slov o bezpečnosti

Tento díl seriálu vás uvede do problematiky bezpečnosti linuxového serveru a pohledů na ni. Jedná se čistě a pouze o obecný pohled na problematiku, nikoliv o přehled konkrétních opatření.

### Úvod

Každý správce serveru by určitě rád zajistil, aby jeho server zůstal co nejlépe zabezpečený, aby nad ním nezískal vládu nějaký útočník, ale také aby server korektně fungoval, byl dostupný a v případě nějaké havárie nebyl velký problém činnost serveru rychle obnovit. Já osobně řadím do oblasti bezpečnosti vše výše uvedené, nikoliv pouze bezpečnost ve smyslu ochrany či prevence proti průniku nějakým útočníkem. Je samozřejmě důležité mít server v bezpečí před nenechavci nebo dokonce před cíleným útokem, ale bezpečnostní strategie by zde neměla končit. Je důležité zabezpečit i samotný běh serveru a připravit se na případné havárie serveru nebo jeho komponenty či chyby softwaru nebo samotného správce.

### Pohledy na bezpečnost

Na bezpečnost lze nahlížet z mnoha hledisek. Já se omezím na dvě, která považuji za klíčová. Především, bezpečnost je vždy kompromisem mezi náklady a úrovní zabezpečení. Čím vyšší úroveň zabezpečení, tím více prostředků, času, práce a peněz je třeba vynaložit. Přitom nikdy není možné dosáhnout stoprocentní bezpečnosti. Té je nanejvýš možné se hodně přiblížit, avšak za cenu astronomických nákladů (a také vedlejších účinků – snížení produktivity firmy kvůli bezpečnostní administrativě, odmítnutí přístupu uživatelům ke službám serveru kvůli příliš agresivním bezpečnostním opatřením atd). Jako správci vlastních serverů si budete požadovanou úroveň zabezpečení volit sami, s tím, že u každého opatření budete zvažovat jeho přínos v porovnání k nákladům na jeho implementaci a údržbu.

Druhý náhled na bezpečnost je jako na proces se čtyřmi fázemi – analýza, implementace, monitorování, změna (či krize). Na počátku probíhá analýza, jejímž výstupem je bezpečnostní strategie, resp. sada konkrétních opatření, politik, úkolů, procesů, odpovědností atd.; uvedení této strategie do provozu představuje druhá fáze, tedy implementace. Následuje patrně ta nejdůležitější fáze vůbec, a sice fáze monitorování a kontroly. V této fázi se prověřuje samotná implementace, zdali byla úspěšná, zdali bylo všechno nastaveno správně, ale také to, zdali jednotlivá pravidla a opatření skutečně fungují a jsou účinná (např. agresivní politika práce s hesly může snížit bezpečnost, pokud si uživatelé budou hesla psát na papírky a lepit na monitor). Druhou věcí, která probíhá v této fázi, je sledování serveru, tzn. sledování logů, sledování vytížení zdrojů serveru, zálohování, kontrola záloh, atd.

Poslední fází je změna nebo také krize, kdy dojde k něčemu, co vás dostane zpět k první fázi – analýze – a celý proces se opakuje. Možná zjistíte, že nějaké opatření bylo příliš agresivní a ve finále odradilo uživatele či zákazníky, nebo je donutilo k něčemu, co bezpečnost ve finále snížilo, takže musíte dané opatření změnit. Nebo vám na server úspěšně pronikne útočník, a na vás je zjistit, jak se mu to povedlo, obnovit činnost serveru a zajistit, aby se útočník na váš server už neproboural.

Bezpečnost je tedy něčím, nad čím je nutno neustále bdít. To je asi to nejdůležitější, co byste si z tohoto článku měli odnést. Představa, že si postavíte a nakonfigurujete server, a poté na něj zapomenete a nebudete se mu již nadále věnovat, je z pohledu bezpečnosti časovaná bomba. A to i v případě, že se jedná o váš soukromý server. I kdyby vám nezáleželo na datech, která na něm máte uložená, stále je tu možnost úspěšného průniku útočníkem, který pak může váš server využít k rozesílání spamu nebo k útoku na jiné počítače v Internetu (nebo ještě hůř – váš server pak může začít sloužit jako skladiště a distribuční uzel pro různé vysoce nelegální materiály).

### Bezpečnost jako proces

Dovolte mi projít jednotlivé fáze procesu zabezpečení a trošku je rozvést. V tomto dílu seriálu nebudu zatím příliš konkrétní, jednotlivé techniky a opatření proberu v některém z dalších dílů. V souvislosti s tím si dovolím upozornit na předchozí díly seriálu, kde se již probíralo něco málo z toho, co může tvořit část vaší bezpečnosti (jmenovitě: diskové šifrování, zabezpečení SSH a základy iptables).

### Analýza

Východiskem pro analýzu by měla být požadovaná úroveň zabezpečení, tzn. zvolený bod na křivce tvořící poměr mezi úrovní zabezpečení a vynaloženými náklady. Asi jinak budete postupovat, bude-li se jednat o váš soukromý server, kde budete mít třeba svůj osobní blog, a jinak v situaci, bude-li se jednat o kritický server pro fungování vaší firmy. S tím bude souviset i to, jaké prostředky máte k dispozici a co jste ochotni do zabezpečení investovat.

Z hlediska opatření proti škodám způsobeným útočníky je třeba uvažovat o dvojím typu útočníka – vnějšího a vnitřního. Vnější útočník je ten, který má přístup pouze k veřejným službám serveru (webové stránky apod.). Vnitřní útočník je ten, který má přístup k některému z uživatelských účtů (nemusí to být nutně zrovna daný uživatel, ale třeba někdo, kdo získal jeho přihlašovací údaje). Zabezpečení vůči vnitřnímu útočníkovi bývá obtížnější (zejména, pokud má daný uživatel přístup k shellu – takový útočník má podstatně více možností, jak škodit nebo jak proniknout hlouběji). Existuje samozřejmě řada zranitelností, které mají za následek „privilege escalation", tedy povýšení oprávnění. Ty pak mohou z vnějšího útočníka učinit útočníka vnitřního. Může to být třeba díra ve webovém serveru, která umožní útočníkovi přístup k shellu pod uživatelem, pod kterým webový server běží. Uvážit je třeba také to, že škodu nepůsobí pouze lidé s úmyslem škodit (útočníci), ale i chybující uživatelé či samotný správce (aneb i mistr tesař se někdy utne). Stačí pouhý překlep při použití známého nástroje rm.

Uvážit je samozřejmě třeba i ostatní oblasti bezpečnosti jako zajištění fungování serveru (redundance, vysoká dostupnost atd.), bezpečnost dat (tzn. jednak strategii pro zálohování a jednak strategii vůči fyzickým útokům na server), aktualizace softwaru, bezpečnostní aktualizace atd. Pokud se na správě serveru podílí více osob, je třeba zajistit jejich koordinaci, popřípadě pravidla bezpečnosti práce atd.

Výstupem této fáze nejsou tedy pouze opatření týkající se daného serveru, ale i příslušných osob, ať již uživatelů, tak správců. V případě firem a produkčních serverů bude analýza podstatně podrobnější a bude obsahovat mj. i scénáře a postupy pro řešení krizových situací.

### Implementace

Tuto fázi asi není třeba příliš rozebírat. Snad jen pokud se jedná o produkční server, který právě běží, vyplatí se maximální opatrnost při konfiguraci služeb (otestovat syntaxi konfiguračních souborů pomocí nějakého nástroje, je-li to možné) či firewallu (aby správce neodstřihl sebe a případné zákazníky). U produkčních serverů nebývá na škodu vyzkoušet si změny nejprve někde nanečisto (virtuální server, testovací server atd.).

### Monitorování

Pokud byste si měli z tohoto dílu něco odnést, pak je to důležitost této fáze. Zde je nejprve třeba otestovat, zdali implementace proběhla správně a zdali všechna bezpečnostní opatření fungují tak, jak mají. To znamená, že byste měli použít příslušné nástroje a přesvědčit se, ať už pokusem o průnik zvenčí nebo zevnitř. Pro firmy je určitě vhodné doporučit bezpečnostní audit nějakým vnějším subjektem. Stejně tak je třeba otestovat systém zálohování, zdali je možné provést obnovu dat (není nic horšího než zjištění, že díky nějaké chybě došlo ke znehodnocení záloh třeba za poslední půlrok, nejlépe v situaci, kdy došlo ke krizi a vy zálohu akutně potřebujete). Funkčnost zálohování, ale i jednotlivých bezpečnostních opatření, je třeba prověřovat pravidelně. Prověřit to jednou rozhodně nestačí.

Součástí této fáze je i běžné sledování serveru, čtení logů a zpráv, zaznamenávání vytížení jednotlivých systémových zdrojů serveru (paměť, disk apod.), sledování výkonu serveru (zdali stíhá v přijatelném čase reagovat na požadavky klientů) atd.

### Změna / krize

Pokud dojde k pouhé změně, resp. nedojde k žádné krizi, pak není příliš o čem hovořit. Zkrátka se vrátíte zpět k analýze, provedete patřičné změny, implementujete je, zkontrolujete a dostanete se zpět do fáze monitorování. Pokud dojde k nějaké krizi, asi tím nejdůležitějším je nepanikařit. Panika má tendenci působit řadu dodatečných škod a problém spíše prohloubit než optimálně vyřešit či alespoň minimalizovat škody. Zachovejte tedy chladnou hlavu a jednejte s rozvahou. Nikde není řečeno, že po zjištění problému musíte problém začít řešit ještě tu danou minutu. Mnohdy se vyplatí chvilku počkat, uklidnit se, popřemýšlet o možných způsobech řešení, popřípadě se podívat do firemní či bezpečnostní dokumentace, jak postupovat v dané situaci.

V případě průniku se může vyplatit věnovat nějaký čas analýze útoku nebo i chování útočníka ještě před samotným odstavením serveru a obnovou ze zálohy – pokud nepřijdete na to, kudy se vám útočník na server dostal, obnova ze zálohy a opětovné rozběhnutí serveru bude mít nejspíše za následek opětovné napadení. Pokud je útočník z České republiky, může se vyplatit sesbírat všechny možné informace kvůli případné žalobě či podání trestního oznámení. V souvislosti s tím nemusí být od věci udělat si kopii napadeného systému a analyzovat ji po vyčištění a obnovení činnosti serveru.

## Správa linuxového serveru: Bezpečnost linuxového serveru

Jakým způsobem zabezpečit linuxový server? Jaké jsou cesty, kterými se útočník může probourat na server? Jakou strategii zvolit při zabezpečení linuxového serveru? Na tyto otázky vám pomůže odpovědět tento a následující díl seriálu.

Navážu na předchozí díl, který se zabýval čistě teorií bezpečnosti, a rozvedu tyto koncepty trošku více do praxe. Nejprve by bylo dobré rozdělit bezpečnost na jednotlivé oblasti, které je třeba zajistit. V případě linuxového serveru přichází v úvahu server jako takový, systém a služby, dostupnost a bezpečnost dat.

### Bezpečnost serveru

Bezpečností serveru je myšlena fyzická bezpečnost, tedy místo, kde je server uložen, kdo má k němu přístup, jak je úložiště serveru zabezpečeno proti vniknutí nežádoucích osob atd. Jelikož v dnešní době začínají být velmi populární virtuální servery, je třeba brát ohled i na tyto typy serverů a jejich úskalí. K serveru má přístup obvykle poskytovatel dané služby, přičemž není vždy jasné, jaké poměry v rámci správy daného serveru panují - kde jsou vaše data, jsou-li zálohována, co se děje se záložními médii, jakým způsobem jsou na nich likvidována data před jejich vyřazením, pokud vůbec, atd.

Kromě toho, fyzický server provozující virtualizaci (hypervizor) používá více uživatelů. Ačkoliv virtualizační technologie poskytují mechanismy k velmi dobré izolaci jednotlivých virtuálních strojů, bezpečnostní díry se najdou všude a nelze vyloučit možnost, že se útočníkovi podaří z napadeného virtuálního stroje probourat do systému hypervizoru, pod kterým vše běží.

K této oblasti nemám příliš mnoho komentářů, spíše jen řadu otázek, na které si musíte sami najít odpověď. Ostatně, bezpečnost je vždy kompromisem mezi časem a prostředky, které do jejího zajištění vložíte, a mezi úrovní bezpečnosti, kterou prostřednictvím této investice dosáhnete. Jinými slovy: v případě osobního serveru, na kterém běží vaše osobní webové stránky, popřípadě nějaký SCM hosting se zdrojovými kódy vašich FOSS projektů, je asi zbytečné investovat do vlastního serveru ve vlastním datacentru a s vlastní ostrahou. A naopak: pokud realizujete prostřednictvím svých serverů své podnikání, a server obsahuje třeba osobní údaje vašich klientů, pak by vám rozhodně nemělo být jedno, v jakém prostředí a v jakých poměrech se server nachází.

### Bezpečnost systému a služeb

Bezpečnost systému a služeb je bezpečnost týkající se náhodných či cílených útoků, ale i omylů uživatele či správce. Kupříkladu, pokud má uživatel přístup k Shellu a zadá obávané `rm -rf /*` (pro jistotu dodávám, že tento příkaz maže celý adresářový strom, a proto byste jej rozhodně neměli spouštět, a už vůbec ne s právy roota!), neměl by ohrozit ani systém, ale ani data jiného uživatele. Toto téma je poměrně rozsáhlé, a proto mu bude vyčleněn celý příští díl seriálu.

### Dostupnost

Jak už jsem nakousl v minulém díle, bezpečnost by neměla končit pouze u bezpečnosti serveru, systému, služeb a dat, ale měla by zohlednit i dostupnost. Pokud bude server delší dobu mimo provoz, pak sice bude v bezpečí proti vzdáleným útočníkům, ale jeho nedostupnost bude přesto působit problém, dokonce možná stejně závažný problém, jako kdyby byl napaden.

Velká část dostupnosti je spojena se samotným hardwarem. Asi tím nejlevnějším řešením je kromě záložního zdroje (UPS), který bude server schopen napájet, i když dojde k výpadku napájení, také nějaká forma RAIDu s redundancí, která zajistí, že výpadek disku nenaruší dostupnost, a dá vám možnost potenciální krizi zažehnat v zárodku. Připomínám, že této tématice se věnovaly tři z mých článků, a sice RAID teoreticky a Softwarový RAID prakticky (první díl, druhý díl). Ostatní možnosti zvýšení dostupnosti zahrnují kvalitnější (a o dost dražší) hardware s redundancí třeba na úrovni zdrojů napájení, připojení k síti, I/O modulů či modulů pro vzdálenou správu serveru (management modulů).

Dalším stupněm mohou být řešení s vysokou dostupností (high availibility), zahrnující obvykle více než jeden server a redundanci na úrovni služeb (vypadne jeden server, požadavky se budou předávat druhému) atd. V této oblasti je mnoho možností, které jsou ovšem vyváženy vysokou cenou, a kromě toho jsou také mimo zaměření tohoto seriálu.

U fyzických serverů nezapomeňte na propojení serveru se záložním zdrojem, aby se váš systém v případě kritického stavu baterie sám bezpečně vypnul.

### Bezpečnost dat

Bezpečnost zahrnuje jak bezpečnost dat vůči zcizení či zneužití nějakým útočníkem, tak bezpečnost dat vůči jejich ztrátě. Se zajištěním bezpečnosti dat vůči zcizení souvisí bezpečnost služeb a systému (útočník, který pronikne do systému, se dostane i k datům, které může zneužít). Stejně tak souvisí s bezpečností serveru (útočník by neměl mít možnost získat fyzický přístup k serveru a třeba si odnést pevný disk). Pokud používáte virtuální server a nemáte pod kontrolou hypervizor, musíte počítat i s únikem dat touto cestou (poskytovatel si může vaše data odnést, nebo hodí pevný disk se starými zálohami do popelnice, kde bude někým objeven atd.).

S bezpečností dat vůči zcizení/zneužití vám může pomoci diskové šifrování. V souvislosti s tím připomínám, že tématice šifrování v GNU/Linuxu jsem se věnoval v třídílném miniseriálu Šifrování s dm-crypt/LUKS (první díl, druhý díl a třetí díl). Připomínám také, že šifrování je vám samozřejmě k ničemu, pokud máte k dispozici virtuální server, jehož hypervizor má útočník pod kontrolou. V takovém případě má přímý přístup k paměti vašeho virtuálního stroje, může si z ní vytáhnout šifrovací klíče a data dešifrovat. Pokud máte k dispozici fyzický server, a útočník k němu získá přístup, když server běží, je vám šifrování opět k ničemu (tzv. cold boot attack, klíč lze získat díky tomu, že data se po výpadku napájení z operační paměti neztratí hned, ale až za nějakou dobu - při podchlazení modulů i za delší dobu). Pro bezpečnost dat je tedy klíčové mít zajištěnu fyzickou bezpečnost serveru, ale také bezpečnost systému a běžících služeb (pokud se útočník proboura třeba přes děravou službu a dostane se k Shellu, získá přístup k datům "zevnitř" systému, kterému jsou data samozřejmě přístupná v nešifrované podobě).

Pokud data nešifrujete, musíte si dávat pozor na uniklé pevné disky, ať již funkční či porouchané - vyhazovat disky do popelnice bez jejich důkladného výmazu není dobrý nápad, dávat je do bazaru je ještě horší, ale stejně tak není úplně ideální vrátit disk výrobci a uplatnit reklamaci (nevíte, kdo dostane disk do ruky, a jak vážná závada to je, tj. v jakém stavu jsou data na porouchaném disku). Výrobci serverů obvykle umožňují za určitý poplatek, aby si majitelé serverů porouchané disky ponechali, takže se nemusí vrátit výrobci na výměnu za nové. Zvažte také, že ačkoliv není možné obnovit všechna data z jednoho disku v diskovém poli (není-li to RAID 1), stále je možné dostat se ke kouskům dat (u RAIDů s prokládáním jako RAID 0, 5 a 6 se data zapisují po blocích, které mají určitou velikost). Dodám, že k důkladnému výmazu dat z pevného disku vám poslouží nástroj shred. Výmaz ovšem nelze provést, pokud se disk porouchá - v takovém případě sice může dojít k částečné likvidaci dat, ale také může odejít pouze kus elektroniky, zatímco data na plotnách zůstanou (i to je třeba důvodem, proč je rozumné citlivá data šifrovat).

#### *Zálohování*

Mít data v bezpečí před zcizením či zneužitím je samozřejmě důležité, ale neméně důležité je mít data v bezpečí před jejich ztrátou. Z tohoto důvodu je nutné data zálohovat. K tomuto účelu existuje v GNU/Linuxu nepřeberné množství nástrojů, od těch nejobyčejnějších nástrojů jako rsync, ssh, tar, dd apod., až po speciální zálohovací nástroje jako například rdiff-backup, duplicity atd.

Co, jak často, jakým způsobem a kam zálohovat, to si musíte určit sami (kromě dat se může hodit zálohovat i systém jako takový - ne celý, pouze jeho klíčová data, tj. adresář /etc s konfigurací, dále pak seznam nainstalovaných balíčků atd.). Co se týká cíle zálohování, bývá dobré zálohovat co nejdále od serveru (nejlépe na jiný server). Zálohovat můžete ale i současně na více úložišť, třeba na externí disk či diskové pole, a také na jiný server. Zálohy mají obvykle dvě podoby - plná a inkrementální (rozdílová). Plná záloha vytvoří kompletní obraz zálohovaných dat (komprimovaný či nekomprimovaný), zatímco u rozdílové se zaznamenají pouze rozdíly mezi současným stavem dat a poslední plnou či rozdílovou zálohou. Plné zálohy bývá dobré provádět tak jednou měsíčně, nebo jednou za delší období, rozdílové zálohy pak denně.

U zálohování je klíčové pravidelně ověřovat stav a funkčnost záloh (nejlépe tak, že se pokusíte něco obnovit)! Není nic horšího, než když vám server selže, vy přijdete o data na něm, a když se podíváte na zálohy, zjistíte, že data z nich nejde obnovit, neboť vám mezi tím došlo místo a vy máte k dispozici data třeba půl roku stará (nebo vůbec žádná, protože zálohovací médium selhalo). I to je jeden z důvodů, proč je dobré zálohovat na více míst než jen na jedno.

Při zálohování musíte dbát také na konzistenci dat. Pokud nějaký proces či uživatel s daty manipuluje, a vy zrovna budete provádět zálohu, získáte nekonzistentní data, která po obnově budou v podobném (nebo ještě horším) stavu, jako kdyby uprostřed práce vypnuli proud (a váš server nebyl na UPS). To se týká zejména databází, u kterých je třeba zálohovat data prostřednictvím databázového systému, rozhodně ne tak, že necháte databázi běžet a zazálohujete její datové soubory, když do nich databázový systém zrovna zapisuje. Možnosti jsou dvě - můžete zálohovat prostřednictvím zálohované služby, umožňuje-li získat konzistentní data, tzn. u databází můžete zálohovat pomocí databázového systému (mysqldump, pg_dump čipg_dumpall atd.). U služeb, které neumožňují konzistentní zálohování za běhu, vám nezbude než službu pozastavit, zazálohovat a znovu spustit (nebo službu provozovat v clusteru s replikací a zálohovat sekundární uzel po jeho odstavení atd.). Určitě si u jednotlivých služeb projděte dokumentaci, zejména pak u zálohování databází. Něco málo k tomuto tématu máte k dispozici v odkazech pod článkem.

## Správa linuxového serveru: Pár slov o útocích na server

Ještě před tím, než se pustím do bezpečnosti systému a služeb, osvětlím, před čím vlastně server chráníte, tedy jaké jsou typy útoků na server, jejich charakter, cíle a motivy útočníků, ale také kudy a jak mohou útočníci na server proniknout a jak mohou škodit.

### Pár slov o útocích na server

Z hlediska směru vedeného útoku uvažujeme o dvou typech útoků - vnějším a vnitřním. Většina potenciálních útočníků bude na server přistupovat z vnějšku, tedy nikoliv jako uživatel daného systému, ale pouze k veřejným, běžícím službám, ke kterým existuje z jejich IP adresy na daný server přístup, tedy např. k webovému serveru, poštovnímu serveru, ale také třeba SSH serveru, pokud je na něj přístup odkudkoliv. Druhým typem útoků je vnitřní, který zahrnuje ostatní případy, tj. situaci, kdy útočník získá přístup třeba k Shellu nebo se proboural přes nějakou běžící službu.

#### *Vnější útoky*

V ideálním případě by k zamezení vnějších útoků měly stačit bezpečnostní mechanismy jednotlivých služeb, ovšem za dvou předpokladů, které jsou nesplnitelné. Prvním předpokladem by byl dokonalý administrátor s kompletní znalostí všech služeb a bez možnosti něco opomenout či udělat chybu, a druhým předpokladem je bezchybné fungování daných služeb. I kdybychom předpokládali neomylnost a stoprocentní znalosti správce, stále tu zůstane fakt, že v každém programu jsou chyby, a řada chyb v serverových službách je využitelná útočníky. Navíc některé služby využívají dalších aplikací či dodatečných modulů, ve kterých mohou být také chyby. Tohle je třeba případ webového serveru, který může být mistrně zabezpečen, ale sám o sobě většinou funguje spíše jako proxy, tj. zachytí požadavek a předá jej interpretu příslušného programovacího jazyka (PHP, Perl, Python, Java apod.), ve kterém běží vlastní webová aplikace, která příslušný požadavek zpracovává.

Řada vnějších útoků je tedy vedena nejenom na služby ve smyslu běžících démonů jako Apache, Postfix, OpenSSH apod., ale i na aplikace, ke kterým některé z daných služeb zprostředkovávají přístup. Tyto aplikace mohou být nakonfigurované buď samotnými správci (lepší případ), nebo také samotnými uživateli (případ webhostingu). To může působit problém - správce obvykle dohlíží na aktuálnost softwaru a záplatování bezpečnostních trhlin (je to v jeho zájmu), ale uživatelé webhostingu jsou různí - někteří na bezpečnost svých aplikací dbají, jiní nainstalují aplikaci a už se o ni nestarají. Vznikají tak nezáplatované, neaktualizované aplikace, které jsou pro daný server potenciální hrozbou.

#### *Vnitřní útoky*

Vnitřní útok je útok útočníka, který má k serveru jiný než vnější přístup. Může získat nebo uhodnout přihlašovací údaje některého uživatele, může to v některých případech být i samotný uživatel (nespokojený zákazník či zaměstnanec), může získat přístup využitím bezpečnostní trhliny v některé z veřejně přístupných aplikací atd. Vnitřní útoky se liší podle toho, kam se útočník dostal. Asi úplně nejhorší je, pokud využije zranitelnost v některé službě, která běží s právy roota, a získá root Shell. To je konec - v takovém případě si může útočník se serverem dělat, co se mu zlíbí. Má navíc široké možnosti svou přítomnost maskovat - nahráním speciálního jaderného modulu či podvržením nástrojů jako ls, netstat, ps, top, ale ibash apod., zajistí, že některé procesy, soubory či uživatele neuvidí ani root.

Druhý nehorší případ nastane, pokud se útočník dostane k uživatelskému Shellu. Nemá sice práva roota, ale to mu obvykle vůbec nebrání škodit jinak nebo se pokoušet dostat dál. Může využít nějaké vnitřní zranitelnosti (nenastavené heslo pro správce databáze atd.) a získat přístup k citlivým datům nebo chráněné vnitřní službě, která není zvnějšku přístupná. Může zkoušet různé zranitelnosti jádra či pod rootem běžících démonů, aby získal root Shell. I uživatelský účet ovšem většině útočníků stačí - mohou na server nainstalovat, který jim umožní začlenit server do botnetu - častý je v takové situaci upravený IRC bot, který se připojí na specifikovaný IRC kanál a čeká na příkazy od útočníka, ale stejně tak může "robot" hledat příkazy od útočníka prostřednictvím protokolu HTTP a některých veřejných služeb jako Twitter apod.

Třetí možností je omezený přístup třeba k interpretu programovacího jazyka apod. - i to někdy stačí k začlenění serveru do botnetu nebo k možnosti postoupit dál k některému z horších scénářů. V této situaci se útočník patrně také dostane, byť třeba pouze omezeně, k datům (přístup k databázi apod.). Útočník samozřejmě může také získat přístup do administračního rozhraní webové aplikace apod. - to sice není až tak kritické z pohledu bezpečnosti serveru, ale z pohledu bezpečnosti služby, kterou server poskytuje, to problém samozřejmě je.

Obecně, vnitřní útoky jsou mnohem závažnější, protože uchránit vnitřek systému bývá obecně mnohem složitější než ochránit systém zvnějšku. Řada služeb nemusí být zvnějšku přístupná, ale zevnitř dané služby přístupné jsou - např. databázový server. Stejně tak má útočník přístup k podrobným informacím o systému, k řadě nástrojů, které jsou na serveru nainstalované, a co je horší, má možnost nainstalovat si na server své nástroje.

#### *Automatizované a cílené útoky*

Většina útoků, se kterými se na serverech potkáte, budou automatizované. Jsou to roboti, kteří na povel útočníka skenují síť a hledají počítače s běžícími službami, přes které se pak snaží získat přístup k Shellu nebo takový přístup, který jim obvykle umožní nainstalovat program, pomocí kterého začlení daný počítač do botnetu. To se dá realizovat i bez přístupu k Shellu, třeba prostřednictvím zranitelnosti ve webové aplikaci. V rámci botnetu pak server rozesílá spam, skenuje nebo útočí na jiné počítače v síti, v horším případě skladuje nějaký ilegální materiál (popř. slouží jako prostředník pro distribuci takového materiálu).

Automatizované útoky jsou sice časté, ale představují relativně menší hrozbu - většina z nich se pokusí najít nějakou triviální zranitelnost, a pokud ji robot nenajde, jde „o dům dál". Pravidelně aktualizovaný software a dobrá politika volby přístupových hesel jako obrana postačí. Horší jsou cílené útoky, kde sice může, ale také nemusí docházet k proniknutí útočníka do systému - pro tuto oblast jsou pak charakteristické DoS a DDoS útoky, a to buď na úrovni služby, kterou jeden nebo více počítačů zahltí požadavky, nebo na úrovni sítě (cílová IP adresa je zahlcena pakety). Cílený útok je nebezpečnější v tom, že za ním obvykle stojí konkrétní útočník nebo útočníci, tedy lidé, a ne automatizované nástroje. V případě napadení serveru zevnitř dokáže šikovný útočník buď získat roota (využitím nějaké zranitelnosti), nebo provést DoS útok na server zevnitř, třeba fork bombou.

#### *Motivy a cíle útoků*

Motivy útoků mohou být samozřejmě různé. Nejčastější je snaha začlenit server do botnetu, a prostřednictvím něj pak páchat různou činnost (rozesílání spamu, DDoS útoky na jiné počítače apod.). V některých případech jsou počítače napadány s cílem jejich „prodeje", resp. prodeje získaného přístupu k danému počítači. Ostatní motivy mohou zahrnovat osobní důvody (msta bývalého zaměstnance či nespokojeného zákazníka, ale i útok nějakého nezúčastněného, kterému se váš server jen tak znelíbil), konkurenční boj, vydírání apod.

Cíle útoků mohou zahrnovat kromě zahlcení serveru (DoS) a začlenění serveru do botnetu i získání cenných dat, které server obsahuje, tzn. databáze zákazníků, osobních údajů, čísel kreditních karet apod. Destruktivní cíle (zničení serveru, smazání určitých dat apod.) jsou méně časté, ale mohou být součástí třeba konkurenčního boje nebo msty. Předpokládám nicméně, že většina čtenářů tohoto seriálu se bude potýkat výhradně s automatizovanými útoky.

## Správa linuxového serveru: Bezpečnost a monitorování systému

Minulý díl se věnoval útokům na servery, jejich charakteru, cílům a motivům. Tento díl se bude věnovat metodám, postupům a nástrojům, které můžete použít k zabezpečení serveru a jeho monitorování.

### Optimální zabezpečení serveru

Na úvod opět zdůrazním, že bezpečnostní opatření jsou kompromisem mezi vynaloženými prostředky a dosažením požadované úrovně zabezpečení. Jinými slovy, opatření je třeba volit s ohledem na to, co chcete chránit, před čím a jaké máte dostupné prostředky. Optimální zabezpečení serveru je tedy takové, které si zvolíte s ohledem na situaci.

Jedním z důležitých předpokladů pro bezpečnost serveru je rozumět systému jako takovému, dále službám, které na serveru provozujete, stejně jako všem bezpečnostním opatřením, které se rozhodnete nasadit. Nasazovat něco, o čem netušíte, jak to vlastně funguje, představuje potenciální riziko. Dostupnost nejrůznějších návodů pro konfiguraci té či oné služby je v tomto případě částečně kontraproduktivní, protože vás nevede k tomu, abyste nejprve nahlédli do oficiální dokumentace.

Ideální je, pokud máte možnost server nakonfigurovat od začátku, tj. pokud dostanete čistý server, na který je třeba systém teprve nainstalovat. V takovém případě doporučuji začít s minimální možnou instalací a instalovat pouze to, co potřebujete. Pokud se dostanete k fungujícímu serveru, proberte jako první krok všechny služby, a ty, které nepotřebujete, ze systému odstraňte.

### Ochrana systému

Prvním předpokladem pro bezpečný systém je jeho pravidelná aktualizace. V případě Debianu můžete použít nástroje jako apticron či apt-listchanges, které vás informují o dostupných aktualizacích, jakmile se objeví, a vy pak můžete systém včas aktualizovat. Samozřejmě je možné zajistit i automatickou aktualizaci, ale bývá dobré aktualizace provádět spíše ručně (v některých případech mohou vyžadovat interakci, což automatický skript nezajistí). V souvislosti s tímto tématem připomínám článek Jak aktualizovat server(y).

#### *Ochrana proti síťovým útokům*

Síťové útoky směřující na server je možné detekovat pomocí nástrojů, které se schovávají pod zkratkou IDS (Intrusion Detection System). Tyto nástroje buď pasivně naslouchají na síti a hledají typické znaky útoků (např. snort), nebo si sami zaberou porty služeb, které nejsou v systému nainstalované, a monitorují přístupy na tyto porty (nástroj portsentry). Tyto nástroje nicméně umí provádět nejenom pasivní monitorování, ale také aktivní, tzn. umí reagovat na události (a třeba zablokovat IP adresu, která provádí port scan). V souvislosti s tím je třeba poukázat na možné otevření okna pro DoS útok, budete-li používat automatizovanou obranu - vždy zvažte, co se stane, pokud útočník bude na server útočit, ale s falešnou zdrojovou IP adresou. Upozorňuji, že nastavení těchto nástrojů může být poměrně komplikované i pro pokročilejšího správce, např. konfigurace vlastních pravidel pro snort.

#### *Ochrana souborů*

Pokud se útočník dostane do systému a třeba získá oprávnění uživatele root, bude se nejspíše snažit upravit klíčové nástroje (Shell, ls, ps, jádro apod.). V takovém případě se hodí nástroje pro ověřování integrity souborů, tedy nástroje jako Tripwire, stealth, aide apod., které umožňují vytvářet databázi zabezpečených kontrolních součtů vybraných souborů, a ty pak ověřovat. K zabezpečení se obvykle používá asymetrického šifrování. V případě použití těchto nástrojů je třeba dbát na správný postup a na možnosti útočníka (útočník může kompromitovat i samotný nástroj, popřípadě jeho databázi).

#### *Ochrana vůči rootkitům*

Rootkit je typický nástroj útočníka, který získá oprávnění uživatele root - je to sada nástrojů sloužící jednak k zajištění pohodlného vzdáleného přístupu ke kompromitovanému systému, ale také k ukrytí své přítomnosti. Některé typické znaky rootkitů mohou odhalit nástroje jako chkrootkit nebo rkhunter. Opět zde však platí to, co bylo řečeno výše - v případě kompromitovaného systému je třeba počítat i s tím, že útočník kompromituje i samotný nástroj pro jeho odhalení.

#### *Zálohování*

Nepředpokládám, že vám musím připomínat, že je třeba provádět pravidelné zálohy. Zmíním snad jen problematiku zálohování systému jako takového. Data obvykle zálohuje skoro každý správce, tam problém nebývá. Systém jako takový už ovšem není zálohován tak často, přitom konfigurace systému i služeb bývá poměrně náročná a při případné ztrátě dat bez dostupnosti záloh systému je třeba ji provést celou znovu od začátku. Proto nebývá od věci zálohovat i systém jako takový. Samozřejmě není třeba zálohovat systém celý včetně všech souborů, zálohovat stačí pouze to podstatné, tzn. adresář /etc s konfigurací systému, dále pak seznam nainstalovaných balíčků a nakonec všechna data. Co se dat týče, nezapomeňte zálohovat opravdu všechna data, např. pokud používáte awstats, pak určitě zálohujte i samotné soubory se statistikami, které se nacházejí ve /var/lib/awstats (některá podobná umístění a data je snadné přehlédnout).

Pokud budete zálohovat Debian nebo na něm založenou distribuci, budete asi chtít zálohovat spíše seznam explicitně nainstalovaných balíčků než seznam všech nainstalovaných balíčků. Pokud používáte nástroj Aptitude, pak víte, že tento nástroj si zapamatuje, co bylo nainstalováno jen jako závislost, a pokud pak odinstalujete původní balíček, Aptitude vám nabídne k odstranění i ty balíčky, které byly nainstalovány jako závislosti. To je velice užitečná funkce z hlediska údržby systému (zejména, pokud na serveru občas experimentujete). Pokud byste zálohovali seznam všech balíčků, zahrnuli byste do toho i ony závislosti a po obnově systému (nebo při jeho migraci jinam) by váš Aptitude tuto svou funkci ztratil. Naštěstí není problém nechat si vypsat explicitně nainstalované balíčky:

```
aptitude -F %p search "?and(?installed,?not(?automatic))" > pkglist.txt
```

A posléze je nainstalovat do nového systému:

```
xargs aptitude --schedule-only install < pkglist.txt
aptitude install
```

### Monitorování systému a služeb

Každý server je třeba monitorovat, tzn. pročítat logy, sledovat různé údaje a jejich změny v čase, abyste včas odhalili případné problémy. Zásadní je sledování logů. To můžete dělat ručně, ale bývá dobré si je nechat zasílat zpracované mailem. Za tímto účelem existují nástroje jako logwatch či logcheck, řízené cronem, které vám jednou za určitou dobu zašlou e-mail se všemi podstatnými událostmi, popřípadě zkonstruují výtah, který se čte o poznání lépe (to zajišťuje třeba logwatch). Tyto nástroje je možné poměrně precizně řídit a filtrovat irelevantní informace, nebo naopak rozšiřovat množství sdělovaných informací.

Sledovat základní údaje jako obsazení paměti, zátěž systému, průchodnost fronty poštovního serveru apod., vám umožní třeba munin ve spolupráci s munin-node. Výstupem je HTML stránka s grafy, které se generují z naměřených hodnot. Z nich můžete vyčíst zdroj nějakých potíží, popřípadě naplánovat, co je třeba na serveru upravit (více operační paměti atd.). Bývá dobré nenechávat výstup nástroje munindostupný z webu alespoň bez nějakého ověření, neboť může obsahovat informace, které mohou být teoreticky využity útočníkem (i když zrovna neobsahuje vyloženě citlivé údaje).

K monitorování teplot můžete použít nástroj lm-sensors, k monitorování teplot pevného disku pak hddtemp. K monitorování zdraví pevných disků můžete použít balíček smartmontools, kterému jsem se již v tomto seriálu věnoval, a to dokonce ve třech dílech. K monitorování softwarového RAIDu můžete použít přímomdadm démona.

Nebývá na škodu ani monitorovat jednotlivé služby, zdali odpovídají, jsou přístupné a dá se s nimi navázat spojení. Toto umožňuje třeba monit či xymon (dříve Hobbit). Nástroj monit umí i reagovat na nedostupnost služeb, zatížení systému nebo obsazení paměti ze strany určité služby, a to upozorněním, restartem či zastavením služby. O všech událostech pak samozřejmě informuje mailem. Takto je možné sledovat nejenom služby samotné, ale i vzdálené servery.

Tím bych tento díl ukončil. Příště se budu věnovat zabezpečení služeb jako takových a zabezpečení sítě, ve které se váš server nachází.

## Správa linuxového serveru: Bezpečnost služeb

Minulý díl se věnoval zabezpečení systému jako takového. Tento díl probere zabezpečení služeb a ochranu sítě, ve které se server nachází.

### Ochrana služeb

#### *Izolace služeb, chroot a virtualizace*

U jednotlivých služeb je především třeba si projít dokumentaci a nastavit je co nejbezpečněji. Nemusí být od věci pokusit se jim omezit přístup jen tam, kam opravdu potřebují. Za tímto účelem je používán nástroj (resp. systémové volání) chroot, kdy se pro službu změní kořenový adresář za nějaký adresář v hierarchii systému (např. adresář s daty). To znamená, pokud konfigurujete webový server s chrootem, pak pro běžící webový server nebude kořenový adresář systémový /, ale třeba /var/www. I kdyby útočník kompromitoval webový server a získal nad ním kontrolu, dostane se pouze k datům ve /var/www, nikoliv pak ke zbytku systému. Konfigurace chrootu nicméně bývá obtížná, zejména tam, kde má aplikace více komponent (např. Apache a PHP), protože do chrootu se pak musí umístit i příslušné binárky, knihovny a v některých případech i speciální zařízení (např. /dev/null apod.). A všechny tyto komponenty je pak samozřejmě třeba pravidelně aktualizovat. Navíc má chroot řadu reálných či potenciálních bezpečnostních problémů. Předně, proces běžící s právy roota se z chrootu může dostat a získat přístup k celému systému (to řeší např. projekt grsecurity, ale ten není oficiální součástí jádra).

Jiné možnosti izolace služeb představuje virtualizace - dnes není až takový problém izolovat služby do několika virtuálních strojů, které běží na jednom fyzickém serveru. Tato forma izolace sice zkomplikuje správu a údržbu takového řešení, ale zase zajistí, že případný útočník, který získá oprávnění roota, kompromituje virtuální server, na kterém běží třeba jen webový server, ale už ne databáze. Nebo, ještě lépe, útočník se probourá na virtuální server, kde běží pouze proxy, která zajišťuje předávání požadavků některému z dalších virtuálních serverů. Dlužno dodat, že i v rámci virtualizace může existovat nějaká zranitelnost, která může za jistých okolností útočníkovi umožnit získat přístup k hypervizoru, tedy mateřskému systému, který virtualizaci řídí a má přístup ke všem virtuálním strojům, i když ta šance je podstatně nižší.

#### *Přístup k síti a síťové útoky*

Pokud není třeba, aby daná služba poslouchala na síti (např. databáze), pak ji nastavte tak, aby naslouchala jen na místní smyčce (127.0.0.1). Někdy se stane, že služba otevírá kromě standardních portů i nějaký další, který třeba slouží pro synchronizaci v clusteru nebo pro správu dané služby, a vy nemůžete nastavit, aby v případě tohoto portu naslouchala jen na místní smyčce. V takovém případě můžete využít firewallu a zablokovat přístup k danému portu (tedy pokud se port při každém spuštění nemění, s čímž už jsem se také setkal - pak pomůže nastavit u firewallu výchozí politiku řetězce INPUT na DROP a naopak jen povolit, co je třeba - toto je ostatně z hlediska stavění firewallu vhodná strategie). Vůči síťovým útokům vám mohou pomoci nástroje jako fail2ban (či denyhosts v případě SSH). Ty mohou zablokovat útočníka, pokud začne páchat brute force útok s různými jmény a hesly ve snaze získat někam přístup. Něco podobného je s jistými omezeními možné vytvořit i na úrovni firewallu. V souvislosti s tím připomínám Praktické rady pro zabezpečení (nejen) SSH. V některých případech lze použít aplikační proxy, ve které můžete precizněji filtrovat typy požadavků na samotnou aplikaci (pokud to jde, je lepší to dělat spíše na úrovni dané služby). Extrémnějším řešením může být skrývání služeb prostřednictvím port knockingu, ale to lze uplatnit spíše u služeb jako SSH, kde není třeba, aby byla daná služba viditelná pro veřejnost.

V neposlední řadě může pomoci i výběr samotných aplikací - některé aplikace mají nepříjemnou historii mnoha bezpečnostních problémů, zatímco jiné byly postaveny s ohledem na bezpečnost, a bezpečnostní problémy u nich nejsou hlášeny tak často. Ze stejných důvodů bývá dobré volit zralé služby, které vyvíjí aktivní komunita, spíše než experimentální služby v rané vývojové fázi, které vyvíjí třeba jediný člověk.

#### *Bezpečnostní patche, SELinux, AppArmor*

Existuje řada oficiálních či méně oficiálních bezpečnostních patchů, které si kladou za cíl zjemnit systém oprávnění tak, aby ani služba běžící s právy roota v případě kompromitace neumožnila postup útočníka do zbytku systému. Zde je nejvíce známý v jádře obsažený SELinux, který je ovšem natolik komplexní, že se ho řada správců štítí, popřípadě ho rovnou vypínají, pokud jej distribuce ve výchozím stavu zapíná. Konkurencí pro SELinux je AppArmor, který je jednodušší, zejména pak na tvorbu nových pravidel (AppArmor to řeší tak, že aplikaci sleduje, zjišťuje, kam přistupuje, a podle toho sám tvoří pravidla přístupu). Dalším zajímavým bezpečnostním patchem je grsecurity, který mj. řeší i bezpečnost chrootu. Tyto mechanismy umožňují zvýšit bezpečnost provozu služeb, ale cenou za jejich použití je nutnost jim dobře porozumět. Jejich nevýhodou je, že špatně nastavená pravidla mohou vést k nefunkčnosti (v lepším případě), nebo k zvláštnímu až takřka "nevysvětlitelnému" chování jednotlivých služeb (v horším případě). Např. SELinux závisí na značkování souborů, takže když jsem jednou přenášel soubory z /var/lib/mysqlpři přesunu dat z jednoho serveru na druhý, přepsal jsem označkované soubory novými, a pak jsem dlouho řešil, proč MySQL tvrdí, že nemá oprávnění přistupovat do tohoto adresáře, když všechna unixová práva jsou nastavena správně. Přirozeně, SELinux umožňuje provádět přeznačkování souborového systému, ale o tom vy musíte vědět a musí vás to napadnout.

### Ochrana sítě

Pokud provozujete třeba domácí server, nebo naopak server v podnikové síti, pak se vás týká nejenom zabezpečení serveru, ale také sítě, ve které se server nachází, a to pokud možno tak, aby případná kompromitace serveru neohrozila jiné počítače v síti. S tím vám může pomoci samotná síťová architektura, respektive tzv. demilitarizovaná zóna. To je koncept, kde figuruje router/firewall se třemi síťovými rozhraními - k jednomu je připojen Internet, ke druhému místní síť a ke třetímu samotný server. Firewall pak řídí přístup tak, aby server mohl přistupovat na Internet (a Internet naopak k němu), ale aby nemohl přistupovat do místní sítě, zatímco místní síť k serveru přistupovat může (viz stavový firewall). Toto řešení je podrobně popsáno třeba v tomto článku. Další mechanismy pro ochranu sítě zahrnují dříve zmíněné IDS, které mohou síť sledovat a upozornit na případné útoky.

### Příliš mnoho bezpečnostních opatření může škodit

Možností pro zvýšení bezpečnosti je mnoho. Já jsem nicméně zastáncem teorie, že příliš mnoho bezpečnostních opatření může ve výsledku buď narušit funkcionalitu či dostupnost, nebo vést paradoxně ke snížení bezpečnosti. Typický příklad snížení bezpečnosti je agresivní politika hesel, která u uživatelů vede k tomu, že si heslo napíší na papírek a nalepí na monitor. Jinou možností může být skutečnost, že každá další služba, kterou na server nasadíte, byť je určena k zajištění bezpečnosti, je místem, kde se může objevit zranitelnost. Pokud tato služba naslouchá přímo na síti nebo monitoruje jakýkoliv uživatelský vstup (viz parsery logů v článku o zabezpečení SSH), je to potenciální problémové místo. Řada aktivních bezpečnostních opatření pak může vést k zablokování přístupu nebo omezení dostupnosti služby. Stejně tak vede příliš mnoho bezpečnostních opatření k tomu, že je lidé (nebo hůř - samotní správci) začnou obcházet či ignorovat a dané bezpečnostní opatření pak ztratí účinnost. Proto je třeba navrhovat bezpečnost rozumně.

## Správa linuxového serveru: OpenVPN server

VPN představuje možnost, jak vytvářet šifrované tunely, kterými je možno směrovat libovolnou komunikaci, počínaje vzdáleným zabezpečeným přístupem k SSH či databázi až po směrování veškeré komunikace přes šifrované spojení. Jak to lze realizovat pomocí OpenVPN, to je předmětem tohoto a následujícího dílu.

### Úvod do OpenVPN

OpenVPN je F/OSS nástroj dostupný snad v každé běžné linuxové distribuci, pomocí kterého je možné vytvářet šifrované tunely mezi dvěma (a více) počítači. OpenVPN může fungovat jako klient nebo jako server, podporuje řadu možností zabezpečení a má širokou škálu nastavení. Z tohoto důvodu bývá obtížné sestavit návod pro zprovoznění OpenVPN, neboť možností, co lze s OpenVPN provádět, je nepřeberně mnoho. V tomto seriálu budou prezentovány dvě alternativy, počínaje tou úplně nejjednodušší s obyčejným sdíleným klíčem, která je předmětem tohoto dílu, a konče pokročilejším řešením s certifikáty, více uživateli a směrováním veškeré komunikace přes šifrovaný tunel, které bude předmětem příštího dílu.

Pokud budete chtít zprovozňovat OpenVPN na svém virtuálním serveru, předesílám, že je nutné mít k dispozici jaderný modul "tun".

### K čemu je to dobré?

VPN může být použito k mnoha účelům, z nichž velmi časté je zabezpečené připojení klientů k intranetu firmy či jiné organizace. Jinou možností je pohodlný zabezpečený přístup ke službám serveru, které jsou zvnějšku nepřístupné (z VPN pak třeba můžete k databázi nebo čemukoliv jinému přistupovat přímo, pokud to ve firewallu povolíte). Obvyklým použitím VPN může být i možnost dostat se zvnějšku do domácí sítě, která je třeba za NATem. Pro úplnost dodávám, že některá použití VPN se kryjí s možnostmi SSH - v rámci SSH je totiž také možné vytvářet šifrované tunely (viz předchozí díly seriálu).

### Jednoduché propojení dvou počítačů

Předesílám, že aby bylo propojení dvou počítačů možné, musí na sebe "vidět". To znamená, že alespoň jeden z nich musí mít veřejnou IP adresu nebo musí být oba ve stejné síti. Pokud jsou oba dva za NATem, ale každý v jiné síti, nebude vám tento návod fungovat. Pokud chcete OpenVPN využívat k přístupu do domácí sítě, která je za NATem, jde to samozřejmě také, ale pouze pokud máte k dispozici nějaký stroj s veřejnou (a pokud možno i statickou) IP adresou. Serverem je vždy stroj s veřejnou a statickou IP adresou. Klientem pak může být i stroj za NATem, na tom nezáleží.

Jako úplně první krok si po instalaci OpenVPN na oba počítače nechte vygenerovat statický klíč, který bude sloužit k zabezpečení vaší komunikace. To můžete učinit následujícím příkazem:

```
openvpn --genkey --secret klic.key
```

Tento klíč je nutné zabezpečeně přenést na druhý počítač. Zde zdůrazňuji slovíčko "zabezpečeně", tedy rozhodně ne přes Internet v nešifrované podobě. Následně je třeba vytvořit dva konfigurační soubory, jeden pro server, jeden pro klienta. Server by měl mít následující konfigurační soubor:

```
dev tun
ifconfig 192.168.15.1 192.168.15.2
secret klic.key
```

Toto je nejjednodušší nastavení, ve kterém jsou zadány pouze nezbytné parametry, přičemž ostatní parametry spojení budou nastaveny dle výchozích hodnot, mj. včetně portu, který má výchozí hodnotu 1194 UDP (tento port musíte povolit na firewallu serveru, alespoň z IP adresy klienta). Parametr `ifconfig` určuje místní a vzdálenou IP adresu v rámci tunelu (v tomto pořadí). To znamená, že server bude mít v tomto tunelu IP adresu 192.168.15.1 a klient 192.168.15.2. Všimněte si parametru secret, který označuje, kde má OpenVPN hledat soubor se statickým klíčem (ideální je zadat absolutní cestu ke klíči - pokud zadáte relativní cestu, musíte OpenVPN spustit ve správném adresáři, jinak soubor s klíčem nenajde).

Parametr `dev` označuje typ zařízení pro virtuální síť. Zde máte dvě možnosti: buď zařízení typu `tun`, nebo zařízení typu `tap`. Zařízení `tun` obaluje protokol IP verze 4 nebo 6 (třetí vrstva OSI modelu), zařízení tappracuje na druhé vrstvě, chová se tedy jako klasický Ethernet.

Konfigurační soubor klienta bude vypadat takto:

```
remote ip_adresa_serveru
dev tun
ifconfig 192.168.15.2 192.168.15.1
secret klic.key
```

Oproti konfiguraci serveru jsou zde dvě změny. Jednak přibyl parametr `remote`, který označuje, s jakým serverem se má OpenVPN klient spojit. Měla by zde být reálná IP adresa serveru. Druhou změnou je obrácení pořadí virtuálních IP adres u parametru `ifconfig`, neboť u klienta je situace přesně opačná, jeho místní adresa je 192.168.15.2 a IP adresa druhého počítače, tedy serveru, je 192.168.15.1.

Nezapomeňte umístit soubor klic.key do správného adresáře. Poté spusťte OpenVPN na serveru:

```
openvpn --config konfiguracni_soubor.conf
```

Pokud se OpenVPN server rozběhne, spusťte OpenVPN stejným způsobem i na klientovi. Za nějakou dobu by obě běžící instance OpenVPN měly ohlásit následující:

```
Initialization Sequence Completed
```

V tuto chvíli by měl být tunel vytvořen a vy můžete zkusit z klienta ping na server:

```
ping -c 1 192.168.15.1
```

Virtuální síť je tvořena příslušným virtuálním síťovým rozhraním na obou počítačích. V tomto případě to bude nejspíše rozhraní tun0. Jeho vlastnosti můžete vypsat následujícím příkazem:

```
ip addr show dev tun0
```

Pokud se cokoliv nedaří, projděte si následující body:

- pokud se nedaří spuštění OpenVPN klienta nebo serveru, přečtěte si chybovou hlášku a pokuste se problém napravit (nejčastější budou asi problémy s cestou ke klíči),
- ověřte dostupnost spojení z klienta na reálnou IP adresu serveru specifikovanou v konfiguračním souboru klienta u parametru `remote` - tato adresa musí být dosažitelná,
- ověřte dostupnost UDP portu 1194 na serveru - port musí být otevřený (zkontrolujte nastavení firewallu příkazem `iptables -L -n -v`),
- ověřte politiku firewallu k nově vzniklým virtuálním síťovým rozhraním (nejspíše tun0) na obou počítačích - pokud máte nastavenou výchozí politiku řetězce INPUT na DROP, přidejte výjimku pro toto rozhraní.

### Komplexnější konfigurace

Pokud vám funguje výše uvedený jednoduchý příklad, můžete se pustit do rozšiřování konfigurace. V první řadě je z bezpečnostních důvodů velmi vhodné nenechávat OpenVPN běžet s právy roota, ale pouze s právy uživatele (např. uživatele nobody). Do obou konfiguračních souborů tedy přidejte následující:

```
user nobody
group nogroup
```

OpenVPN je možné spouštět jako démona (tzn. spustí se na pozadí a hlášky bude vypisovat do logů místo na obrazovku). Toto chování zajistí následující parametr:

```
daemon
```

Debian považuje libovolný soubor s koncovkou .conf v adresáři /etc/openvpn jako konfigurační soubor, pro který spustí samostatnou instanci OpenVPN při startu systému nebo příslušné operaci se startovacím skriptem /etc/init.d/openvpn. Výše zmíněný parametr daemon není v případě těchto konfiguračních souborů nutností (příslušná instance OpenVPN bude automaticky spuštěna jako démon).

Pokud nechcete OpenVPN provozovat na standardním portu, můžete na serveru explicitně udat port, na kterém chcete, aby naslouchal:

```
port 39993
```

Na klientovi je pak třeba upravit parametr remote, kam za IP adresu přidáte číslo portu, tedy např:

```
remote ip_adresa_serveru 39993
```

Bývá také dobré zajistit, aby si obě strany patřičně hlídaly propustnost tunelu a v případě potíží tunel shodili a začali se pokoušet jej znovu vytvořit. To zajistí následující parametry:

```
keepalive 10 60
ping-timer-rem
```

```
persist-tun
persist-key
```

Volba keepalive má dva parametry, které pracují následovně. Tím prvním je doba v sekundách, za kterou se pošle na druhou stranu tunelu "ping", je-li spojení v nečinnosti. Pokud tedy OpenVPN neobdrží žádný paket z druhé strany po tuto dobu (v příkladu výše by to bylo deset sekund), pošle na druhou stranu "ping". Druhým parametrem je "timeout", tzn. doba, za kterou se OpenVPN začne pokoušet spojení restartovat, pokud neobdrží žádnou odezvu z druhé strany (tzn. buď "ping", nebo jakýkoliv jiný paket).

Pokud se OpenVPN pokusí spojení restartovat (stejného výsledku lze dosáhnout ručně zasláním signáluSIGUSR1 běžícímu procesu OpenVPN), nemusí už mít potřebná oprávnění k vytvoření virtuálního síťového rozhraní či k opětovnému přečtení klíče. Z tohoto důvodu lze OpenVPN instruovat, aby zařízení ani klíč nezahazoval, ale nechal je "přežít" restart - to zajišťují volby persist-tun a persist-key.

Poslední volbou zde je ping-timer-rem, který u serveru zařídí, že nebude brát v úvahu parametry volbykeepalive v případě, kdy není připojený žádný klient.

Poslední důležitou volbou je volba zajišťující kompresi dat, která sníží objem přenášených dat za cenu vyšší zátěže procesoru (v tomto ohledu je třeba dodat, že samotné šifrování také představuje zátěž pro procesor, zejména v případě širokého přenosového pásma). Zapnutí komprese zajistí následující volba:

```
comp-lzo
```

V příštím dílu seriálu vám představím komplexnější řešení s certifikáty, s více klienty a se směrováním veškeré komunikace přes VPN.

## Správa linuxového serveru: OpenVPN server (pokračování)

Minulý díl se zabýval základy OpenVPN a velmi jednoduchou konfigurací propojení dvou počítačů. Tento díl osvětlí komplexnější řešení pro OpenVPN server s více klienty a možnost směrování veškeré komunikace přes VPN.

### Komplexní řešení pro více klientů

Vytvoření tunelu mezi dvěma počítači je jistě užitečné, ale co když se chcete připojovat na VPN server z více počítačů, nebo dokonce nabídnout klientům možnost směrovat veškerou svou komunikaci přes VPN? I to samozřejmě OpenVPN umí. Pro snadnou správu přístupu mnoha klientů k VPN serveru využívá OpenVPN PKI. Je tedy třeba začít vytvořením certifikační autority, dále certifikátu serveru a certifikátu klientů.

#### *Vytvoření certifikační autority*

Prvním krokem je vytvoření vlastní certifikační autority. Můžete tak učinit buď ručně pomocí nástroje `openssl`, nebo si značně usnadnit práci pomocí pomocných skriptů. V Debianu naleznete příslušné skripty v adresáři `/usr/share/doc/openvpn/examples/easy-rsa`, takže si je odtud zkopírujte někam jinam, třeba do `/etc/openvpn`, takto:

```
cd /etc/openvpn
cp -r /usr/share/doc/openvpn/examples/easy-rsa .
cd easy-rsa/2.0
```

Postup pro vytvoření certifikační autority je následující. Upozorňuji, že tento postup (přesněji druhý příkaz) smaže všechny klíče, které jste dosud vytvořili (pokud toto provádíte poprvé, není se samozřejmě čeho obávat).

```
. ./vars
./clean-all
./build-ca
```

Po dokončení této procedury byste měli nalézt příslušné výstupy, tzn. zejména soubor `ca.crt` s kořenovým certifikátem certifikační autority a soubor `ca.key` se soukromým klíčem certifikační autority (tím se pak podepisují jednotlivé dílčí certifikáty).

#### *Vytvoření serverového klíče*

Máte-li vygenerovanou certifikační autoritu, můžete si nechat vygenerovat serverový certifikát. To provedete ve stejném adresáři následujícím příkazem:

```
./build-key-server server
```

Za slovo `server` si dosaďte libovolné jméno. Zde budete opět vyzváni k zadání řady informací. Ujistěte se, že na konci odpovíte na dotaz "y", tedy opravdu podepsat daný certifikát.

#### *Vygenerování certifikátů a klíčů pro uživatele*

Pro každého uživatele vygenerujte jeho vlastní certifikát spolu s klíčem, pomocí kterých se bude připojovat k serveru. K tomu vám poslouží následující příkaz:

```
./build-key jmeno_klienta
```

#### *Vygenerování Diffie Hellman parametrů*

Posledním krokem je vygenerování parametrů pro Diffie Hellman protokol. To provedete příkazem:

```
./build-dh
```

#### *Umístění jednotlivých souborů*

Až budete mít vygenerované všechny soubory, bude potřeba je vhodně rozmístit. Obecně, soubory s příponami `.key` obsahují privátní klíče a měly by být pečlivě střeženy (doporučuji jim nastavit práva 400), ostatní být střeženy nemusí. Přípona `.crt` označuje jednotlivé certifikáty - certifikát certifikační autority, certifikát serveru a certifikáty klientů. Diffie Hellman parametry se pak nacházejí v souboru `dh*.pem`, kde hvězdička bude odpovídat velikosti klíče - výchozí hodnota je 1024, ale je samozřejmě možné vygenerovat větší.

Soubory pro klienta

Klienti by měli mít k dispozici následující soubory:

- `ca.crt` - kořenový certifikát certifikační autority
- `klient1.crt` - odpovídající certifikát klienta
- `klient1.key` - odpovídající privátní klíč klienta

Soubory pro server

Server musí mít k dispozici následující soubory:

- `ca.crt` - viz výše
- `server.crt` - serverový certifikát
- `server.key` - soukromý klíč k serverovému certifikátu
- `dh*.pem` - Diffie Hellman parametry

Ostatní soubory

Jediným souborem, který jste vygenerovali a nebyl popsán výše, je soubor `ca.key` s klíčem pro kořenový certifikát certifikační autority. Ten není nutné mít přímo na serveru - je totiž nutný pouze k podepisování certifikátů. Z bezpečnostních důvodů je dokonce vhodné, aby byl někde v bezpečí mimo server.

#### *Konfigurace OpenVPN serveru*

Příklad konfigurace serveru může vypadat následovně. Konfiguraci si přizpůsobte svým potřebám:

```
local 1.2.3.4
dev tap
mode server
ifconfig 10.45.42.1 255.255.255.0
ifconfig-pool 10.45.42.2 10.45.42.102 255.255.255.0
tls-server
ca /etc/openvpn/ca.crt
cert /etc/openvpn/server.crt
key /etc/openvpn/server.key
dh /etc/openvpn/dh1024.pem
user nobody
group nogroup
comp-lzo
keepalive 10 20
ping-timer-rem
persist-tun
persist-key
verb 3
duplicate-cn
client-to-client
max-clients 100
```

U všech cest k souborům se ujistěte, že ukazují správně. Parametr `local` označuje IP adresu serveru, kde bude k dispozici OpenVPN server. Parametr `ifconfig` označuje IP adresu serveru v rámci VPN tunelu. Klientům budou přiřazovány IP adresy z rozsahu, který je specifikován parametrem `ifconfig-pool`. Mezi další důležité parametry patří:

- `tls-server` - zapíná TLS v režimu serveru
- `duplicate-cn` - umožní, aby se připojilo více klientů se stejným certifikátem
- `client-to-client` - umožní klientům, aby na sebe "viděli" a mohli spolu komunikovat

- max-clients 100 - limit počtu současně připojených klientů

***Konfigurace klienta***

Pro klienty použijte následující vzor konfiguračního souboru:

```
remote 1.2.3.4
client
dev tap
tls-client
ns-cert-type server
ca /etc/openvpn/ca.crt
cert /etc/openvpn/klient1.crt
key /etc/openvpn/klient1.key
nobind
comp-lzo
keepalive 10 20
ping-timer-rem
persist-tun
persist-key
verb 3
```

Řadu parametrů zde již jistě znáte, pro jistotu ještě zopakuji upozornění, abyste si přizpůsobili veřejnou IP adresu serveru (parametr remote) a cesty k jednotlivým souborům. Mezi důležité volby patří:

- tls-client - zapíná TLS v režimu klienta
- ns-cert-type server - ověřuje serverový certifikát (pomáhá zabránit man-in-the-middle útokům, kdy by se útočník vydával za server)

***Extra zabezpečení pomocí sdíleného klíče***

Ačkoliv infrastruktura založená na certifikátech nevyžaduje použití sdílených klíčů (což je ostatně jejím účelem), můžete pro zvýšení bezpečnosti sdílený klíč přeci jen vytvořit a používat jako bezpečnostní vrstvu navíc - server pak bude kontrolovat HMAC podpisy paketů a bude ignorovat pakety, které nebudou mít správný podpis. Klíč vytvoříte následujícím příkazem:

```
openvpn --genkey --secret klic
```

Následně je třeba dopravit klíč všem klientům a upravit konfiguraci serveru přidáním následující řádky do konfiguračního souboru:

```
tls-auth soubor_s_klicem 0
```

Nezapomeňte doplnit cestu k souboru s klíčem. Do konfiguračních souborů klientů je třeba doplnit podobnou řádku, ale s číslem 1 místo nuly:

```
tls-auth soubor_s_klicem 1
```

**Směrování komunikace přes VPN**

OpenVPN umožňuje, aby klienti směrovali veškerou svou komunikaci přes VPN tunel, což umožňuje vytvářet různé "anonymizační" služby založené na této technologii. Tato vlastnost se hodí mj. i k tomu, abyste zabezpečili svou komunikaci, pokud se třeba připojujete přes nešifrovanou wifi, kde vás může kdokoliv ve vaší blízkosti šmírovat (alespoň vaši nezabezpečenou komunikaci).

***Směrování paketů***

Abyste zajistili směrování komunikace v rámci OpenVPN serveru, musíte nejprve povolit směrování paketů. To můžete provést buď s dobou trvání do příštího restartu, nebo permanentně. Dočasné nastavení provedete příkazem:

```
sysctl -w net.ipv4.ip_forward=1
```

Permanentní nastavení provedete zapsáním níže uvedené řádky do souboru /etc/sysctl.conf:

```
net.ipv4.ip_forward=1
```

Podívejte se na nastavení firewallu (iptables -L -n -v), zejména pak na řetěz FORWARD, který musí povolovat průchozí pakety z virtuálního rozhraní VPNky směrem na vnější síťové rozhraní.

***NAT / maškaráda***

Jelikož VPN síť využívá adresy z privátního rozsahu, je nutné vytvořit maškarádu, tedy vhodnou formu NATu (překladu adres), která zajistí, aby si pakety zvenčí našly cestu k příslušnému klientovi, který si je vyžádal. To zajistíte následujícím příkazem:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Toto nastavení bude fungovat pouze do příštího restartu serveru. Permanentní nastavení je v Debianu možné zařídit pomocí balíčku iptables-persistent a nástroje iptables-save:

```
iptables-save > /etc/iptables/rules
```

Tento příkaz uloží aktuální konfiguraci pravidel Netfiltru a při příštím spuštění systému bude zajištěno jejich opětovné načtení.

***Úprava konfigurace OpenVPN serveru***

Následně upravte konfigurační soubor OpenVPN serveru a přidejte do něj následující řádky:

```
push "route-gateway 10.45.42.1"
push "redirect-gateway def1"
push "dhcp-option DNS 10.45.42.1"
```

Jako parametr pro volbu route-gateway zadejte IP adresu serveru v rámci VPN, tedy IP adresu, kterou jste zadali jako parametr volby ifconfig. První řádka zajistí, že se klientovi pošle specifikovaná IP adresa brány. Druhá volba zajistí, že se klientovi nařídí, aby přesměroval veškerou svou komunikaci přes VPN. Parametrdef1 zajistí, aby se neodstranila původní brána z konfigurace sítě klienta.

Parametr dhcp-option DNS zajišťuje zaslání informace o DNS serverech klientovi. Tento parametr samozřejmě předpokládá dostupný funkční resolvující DNS server. Pokud se vám nechce nastavovat vlastní DNS server na serveru s OpenVPN, můžete použít buď OpenDNS, nebo Google DNS. Stavba vlastního DNS serveru je mimo zaměření tohoto článku, ale mohu vám v této souvislosti pro tuto konkrétní potřebu doporučit balíček dnsmasq.

***Úprava konfigurace OpenVPN klientů***

U linuxových OpenVPN klientů se ovšem nastavení DNS pomocí dhcp-option neprojeví - je nutné jej odchytit z proměnných prostředí a zpracovat skriptem. Tento skript si naštěstí nemusíte psát sami - měl by být součástí OpenVPN balíku a měl by sídlit v /etc/openvpn/update-resolv-conf. Ke své činnosti vyžaduje balík resolvconf, takže se raději ujistěte, že jej máte nainstalovaný. Pak už jen postačí upravit konfiguraci klienta přidáním následujících řádků do konfiguračního souboru OpenVPN:

```
up /etc/openvpn/update-resolv-conf
down /etc/openvpn/update-resolv-conf
script-security 2
```

***Neposlušný klient aneb Jak si nenechat mluvit do směrování***

Pokud máte takto nastavený server a jste v pozici klienta, který nechce, aby se mu veškerá komunikace směrovala přes VPN, ale vytvořit VPN tunel chcete, můžete do svého konfiguračního souboru přidat parametr route-nopull. Tento parametr způsobí, že OpenVPN nebude ze serveru přebírat informace pro směrování.

## Správa linuxového serveru: Úvod do zálohování

Zálohování je jedna z nejkritičtějších oblastí správy serverů. Je třeba vhodně zvolit, co bude zálohováno, kam, kdy, jak často a jakým způsobem. S tím se vám pokusí pomoci tento článek.

### Úvod

Ztráta dat bývá v oblasti správy serverů jedním z nejfatálnějších problémů. Ačkoliv se můžeme snažit pravděpodobnost ztráty minimalizovat, nikdy nelze zcela eliminovat její riziko. S tím je svázána často přehnaná důvěra k diskovým polím, které, jak už bylo v tomto seriálu několikrát zmíněno, nemohou být náhradou za pravidelné zálohování, neboť ke ztrátě dat může dojít nejenom vlivem selhání pevného disku, ale také následkem omylu uživatele či správce, činnosti útočníka, softwarové nebo hardwarové chyby atd.

Ztráta dat ale není jediným problémem. Kromě samotných dat je třeba uvažovat i o softwarové infrastruktuře, která je s nimi propojena. Tím mám na mysli nakonfigurovaný server s řadou služeb, které k datům přistupují a manipulují s nimi. Do katastrofických scénářů je tak třeba přidat i situace, kdy sice nedojde ke ztrátě samotných dat, ale kdy dojde k omezení fungování serveru (např. ztráta funkčnosti důležité služby vlivem chyby při povýšení na novou verzi distribuce atd.) či ztrátě dlouho vylepšované konfigurace. Zálohovat je tedy třeba tak, aby bylo možné a snadné obnovit nejenom samotná data, ale také služby, které k nim zprostředkovávají přístup.

### Co zálohovat?

Je možné, i když možná trochu nepraktické, zálohovat celý server se všemi soubory, včetně softwaru. Jinou, úspornější možností je zálohovat selektivně jen to, co nelze opatřit jiným způsobem. Mezi obvyklé typy dat a umístění vhodných pro zálohování patří:

- domovské adresáře uživatelů (obvykle v `/home`)
- pošta uživatelů (standardní úložiště se nachází ve `/var/mail`, ale velmi často je pošta uložena někde jinde)
- obsah webových stránek (obvykle se nachází ve `/var/www` či `/srv/http` atd.)
- data z databází - soubory s databázemi a tabulkami nelze zálohovat konzistentně, pokud server běží; podívejte se do dokumentace DBMS, jak nejlépe data zálohovat
- data ostatních běžících služeb (např. Jabber/XMPP server atd.)
- seznam nainstalovaného softwaru - využijte správce balíčků k získání seznamu explicitně nainstalovaných balíčků
- extra software, který byl nainstalovaný mimo balíčkovací systém (obvykle dostupný v `/opt`)
- konfigurace systému a systémových služeb (většina by měla být v `/etc`, ale výjimky existují)
- logy systému a služeb - není to nutné, ale v některých případech to může pomoci (např. při analýze průniku)
- databáze různých monitorovacích nástrojů a nástrojů tvořících statistiky (např. Awstats s daty ve`/var/lib/awstats` atd.)
- atd.

Pokud si nejste jisti, zdali jste postihli úplně vše, můžete zkusit z vytvořených záloh obnovit celý server třeba na nějakém testovacím počítači (nebo ve virtuálním stroji). To bych u produkčního serveru doporučil i v případě, kdy jste si naprosto jisti, že zálohujete opravdu vše potřebné. Zjistíte tak totiž nejenom případná chybějící data, ale také různá úskalí, která by vás normálně nenapadla, stejně jako dobu a náročnost obnovy ze záloh po katastrofě.

### Kam zálohovat?

Zálohovat je možné do jiného adresáře, na externí disk, na externí diskové pole, na jiný server atd., možností je mnoho. Ideální je zálohovat na více než jedno úložiště, pokud možno ještě geograficky odloučené (to pak pomáhá v situaci, kdy případná katastrofa postihne nejenom server, ale třeba i celé datacentrum - viz záplavy, požár apod.). Naopak, mít jedinou dostupnou zálohu v dedikovaném adresáři na stejném diskovém poli, kde jsou data i systém, je velmi špatný nápad (viz výše).

### Jak zálohovat?

Zálohovat je třeba především s ohledem na konzistenci dat, tzn. nezálohovat soubory, do kterých může být právě zapisováno (nejčastěji se jedná o datové soubory databáze).

#### *Rozdílová versus plná záloha*

Zálohovací systémy mají obvykle nejenom schopnost zálohovat všechna data tak, jak jsou (tomu se říká plná záloha), ale i možnost zálohovat pouze změny, ke kterým došlo od předchozí zálohy (tomu se říká rozdílová záloha). Rozdílové zálohování umožňuje pracovat poměrně efektivně s úložištěm pro zálohy a zbytečně na něm neplýtvat místo. Rozdílové zálohování ale skýtá potenciální problém - každá rozdílová záloha je pevně spjata s plnou zálohou. Jak se data mění, rozdílů přibývá, čímž klesá výkon zálohování a celá záloha zastarává. Z tohoto důvodu je vhodné provést jednou za čas plnou zálohu, stejně jako dbát na odstranění starých a nepotřebných záloh.

### Jak často zálohovat?

Volba periody pro zálohování se odvíjí od toho, jak stará data můžete v zálohách mít. Kupříkladu, pokud budete zálohovat denně, budete mít data stará nanejvýš 24 hodin. Pokud budete zálohovat každou hodinu, budete mít data stará nanejvýš hodinu atd. Zde záleží také na tom, o jaká data se jedná - u některých dat si můžete dovolit provádět zálohu jednou za delší dobu, u jiných dat budete zálohovat častěji, protože se častěji mění nebo je pro vás zásadnější mít je v případě katastrofy na zálohách co nejaktuálnější.

Pokud se jedná o typ zálohy (plná či rozdílová), pak plnou zálohu se obvykle doporučuje provádět jednou za delší období, podle okolností třeba měsíc nebo týden. Rozdílové zálohování je pak možné provádět mnohem častěji, třeba denně.

Zálohujete-li na více úložišť, je vhodné provádět plnou zálohu se stejnou periodou u obou úložišť, ale u každého úložiště ji provést jindy, nejlépe tak, aby mezi plnou zálohou jednoho a druhého úložiště byla stejná doba, tzn. provádí-li se na každém úložišti plná záloha jednou za měsíc, pak na jedno úložiště se plná záloha provede na začátku měsíce a na druhé v polovině měsíce.

### Jak dlouho držet zálohy?

Toto je samozřejmě ryze individuální otázka - v některých případech je vhodné zálohy nelikvidovat vůbec, ale pro jistotu (nebo ze zákona) je archivovat, jindy postačí držet zálohy třeba pouze několik týdnů zpět. V případě některých specifických a málo pravděpodobných problémů, např. v případě tichého poškozování dat (data corruption), nemusí být držení záloh několik týdnů zpátky postačující (než se na to přijde, zálohy mohou být již celé poškozeny). Naopak pro většinu situací úplně postačí mít k dispozici jen poslední zálohu. Obvykle se to, jak dlouho je vhodné držet zálohy, odvíjí od kapacity úložiště.

### Čím zálohovat

V tomto seriálu budou počínaje příštím dílem představeny tři zálohovací nástroje, `rsync`, `rdiff-backup` a`duplicity`. Zálohovacích nástrojů ale existuje nepoměrně více, ať už se jedná o FOSS či komerční řešení. Použité nástroje se samozřejmě také odvíjí od toho, kde jsou data uložena. Kupříkladu, disková pole, LVM či virtualizační řešení umožňují provádět snapshoty, což je určitě jeden ze zajímavých způsobů zálohování, byť asi ne zcela postačující, pokud byste kromě tohoto způsobu nezálohovali jinak.

### Bezpečnost záloh

Samotné zálohování ke zmírnění následků katastrof nestačí. Je třeba si uvědomit, že v případě katastrofálního selhání serveru a ztráty dat na něm je záloha, kterou máme někde bokem, už posledním místem, které obsahuje naše cenná data. Jednosměrnost zálohování svádí k tomu, abychom ze zálohovacího média četli až v případě nutnosti obnovit data. Jenomže každé úložiště dat se může pokazit, a co je horší, může se pokazit tak, že zápis na něj sice bude možný, ale opětovné přečtení zálohovaných dat již možné nebude. Proto je třeba zálohy pravidelně prověřovat, v lepším případě ještě zálohovat na více míst.

Kromě toho je samozřejmě třeba uvažovat o bezpečnosti záloh i vzhledem k nejrůznějším útočníkům, kteří by se k vašim datům rádi dostali - data na serveru si správce obvykle důmyslně chrání, ale zálohy mu někdy v uvažování o bezpečnosti dat mohou utéct (nešifrované zálohy s daty ze zašifrovaných svazků, uložení záloh na stole v kanceláři atd.). Spolu s tím je také třeba přemýšlet nad tím, co se stane se zálohami v případě likvidace nebo vyřazení zálohovacích médií (aneb média se zálohami v popelnici nebo třeba v servisu na reklamaci).

## Správa linuxového serveru: Jednoduché zálohování pomocí rdiff-backup

V tomto dílu vám ukáži, jak využít nástroje rdiff-backup k zálohování dat a k případné obnově.

### Úvod

Nástroj rdiff-backup je napsaný v Pythonu, umí rozdílové zálohování a umí pracovat jak lokálně, tak vzdáleně. Unikátní vlastností tohoto nástroje je podoba zálohy. Nástroj totiž vytváří de facto „zrcadlový" obraz původních dat, který uchovává v aktuální podobě, zatímco rozdíly si zaznamenává do speciálního adresáře. To znamená, že máte vždy k dispozici aktuální verze souborů přístupné přímo z cílového souborového systému, a to bez nutnosti použít nástroj rdiff-backup. K obnovení dat z poslední zálohy tak stačí pouhé překopírování. Oproti jiným nástrojům (např. duplicity) naopak rdiff-backup neumí ani kompresi dat, ani šifrování. Jediné, co se komprimuje, jsou samotné rozdíly.

Ačkoliv je rdiff-backup schopen pracovat vzdáleně, doporučuje se, je-li to možné, použít stejnou verzi nástroje na obou počítačích. V případě nesouladu verzí vypíše nástroj varování.

Rdiff-backup umí využívat přenosové pásmo podobně efektivně jako nástroj rsync (tomuto nástroji se věnuje článek Zálohuj svá data tolikrát, kolikrát je máš rád), takže při následných zálohách se vždy přenáší pouze rozdíly a nikoliv znovu veškerá data.

### Archfs

Nasazení nástroje rdiff-backup může vhodně doplnit FUSE modul archfs, který umí připojit adresář se zálohami jako souborový systém, přes který je možné snadno prohlížet data jednotlivých záloh (ty jsou pak dostupné jako adresáře, jejichž název odpovídá datu pořízení). Bohužel tento FUSE modul není k dispozici v oficiálních repozitářích Debianu pro vydání Lenny ani Squeeze, který je momentálně na spadnutí. K dispozici je až ve větvi Sid (unstable), kterou ovšem rozhodně nelze doporučit k používání na serveru.

### Místní záloha

Nejjednodušší příklad použití nástroje rdiff-backup je místní záloha, tzn. záloha v rámci jednoho počítače. V takovém případě je už z principu velmi vhodné zálohovat na jiné médium, než na kterém se nachází zdrojová data. Pokud by byla data k zálohování v adresáři /home/data a připojený externí disk v /mnt/extdisk, záloha by se provedla takto:

```
rdiff-backup /home/data /mnt/extdisk
```

Každé provedení tohoto příkazu způsobí aktualizaci souborů zálohy do stejného stavu jako ve zdrojovém adresáři, přičemž informace o rozdílech se ukládají do speciálního adresáře pojmenovaného rdiff-backup-data.

Ne vždy je ale žádoucí zálohovat veškerá data ve zdrojovém adresáři. Proto má nástroj rdiff-backupbohaté možnosti pro specifikaci dat, která nemají být zálohována ze zdrojového adresáře. Nejjednodušší metodou je použít parametr --exclude:

```
rdiff-backup --exclude /home/data/vyradit /home/data /mnt/extdisk
```

Všimněte si, že parametry a volby předcházejí specifikaci zdroje a cíle – toto pořadí je nutné dodržovat. Potřebujete-li vyřadit více souborů či adresářů, použijte parametr --exclude vícekrát za sebou:

```
rdiff-backup --exclude /home/data/vyradit1 --exclude /home/data/vyradit2 /home/data /mnt/extdisk
```

Možné je i vyřadit určité typy souborů, třeba zařízení (device files), symbolické linky, sokety apod., např. takto:

```
rdiff-backup --exclude-device-files /home/chroot /mnt/extdisk
```

Výše uvedený příkaz nebude zálohovat soubory zařízení (device files). Rdiff-backup nabízí řadu dalších možností filtrování souborů včetně regulárních výrazů. Ty naleznete popsané v manuálové stránce.

### Vzdálená záloha

Vzdálené zálohování je řešeno prostřednictvím SSH. Jak již bylo řečeno, dalším předpokladem je dostupnost nástroje rdiff-backup na cílovém počítači, pokud možno ve stejné verzi jako na zdrojovém. Zálohování na vzdálený počítač pak může vypadat třeba takto:

```
rdiff-backup /home/data uzivatel@server::/cilovy/adresar
```

Problémem, na který můžete při vzdáleném zálohování narazit, je problém s metadaty souborů, tzn. s vlastnictvím a oprávněními. Dojde k tomu třeba v situaci, kdy zálohujete data vlastněná uživatelem, který na cílovém systému neexistuje, nebo třeba v situaci, kdy zálohujete data vlastněná uživatelem root, zatímco na cílový systém se hlásíte s právy obyčejného uživatele. Rdiff-backup si naštěstí metadata zapisuje zvlášť, takže pokud k obnově použijete právě tento nástroj, informace o vlastnictví a oprávněních se přenese korektně. Pokud byste ale obnovovali prostým překopírováním, data by se obnovila se špatnými právy, vlastníkem či skupinou.

### Obnova dat

Pokud dojde k nějakému problému a potřebujete provést obnovu, můžete aktuální data rovnou zkopírovat (pozor ale na problém s metadaty souborů – viz výše) nebo použít příkaz níže, kde na pozici zdroje uvedete zálohu a na pozici cíle adresář, kam se mají data obnovit. Nutné je také specifikovat čas, ke kterému budou data ze záloh obnovena, pomocí parametru -r. Pokud vám stačí nejnovější data, můžete použít časovou značku now, takto:

```
rdiff-backup -r now uzivatel@server::/zaloha /tmp/obnova
```

Jelikož rdiff-backup provádí rozdílové zálohování, můžete obnovit i stav k určitému datu nebo stav k určité provedené záloze. Můžete také specifikovat relativní čas (např. stav před dvaceti dny), takto:

```
rdiff-backup -r 20D /zaloha /tmp/obnova
```

Stav k určitému datu můžete obnovit takto:

```
rdiff-backup -r 2010-12-30 /zaloha /tmp/obnova
```

Je také možné obnovit stav k určité provedené záloze, tzn. chcete-li obnovit stav k sedmé nejnovější záloze, použijete následující příkaz:

```
rdiff-backup -r 7B /zaloha /tmp/obnova
```

Vývojáři nástroje rdiff-backup se snaží, aby bylo jeho použití bezpečné. Proto vám tento nástroj standardně neumožní např. přepsat existující data při obnově či jiné potenciálně rizikové operace. Pokud jste si opravdu jisti, že chcete danou operaci provést, i když vás nástroj tímto způsobem varuje, můžete použít parametr --force.

### Zajištění pravidelného zálohování

Abyste zajistili pravidelné zálohování, postačí přidat příslušný příkaz do cronu, popřípadě si vytvořit skript, přidat mu právo ke spuštění a umístit jej do vhodného adresáře (pro spouštění každý den by to byl adresář/etc/cron.daily). Pokud by v rámci běhu nástroje došlo k chybě, měl by cron informovat uživatele root mailem.

### Čištění úložiště od starých záloh

Jelikož zálohovaná data se časem nahromadí a začnou zabírat nemalý prostor, je dobré vědět, jakým způsobem stará data bezpečně odmazat. K tomuto slouží parametr --remove-older-than, jehož hodnotou je buď absolutní, nebo relativní čas. Drobným problémem je, že v jednom volání nástroje rdiff-backup není možné současně mazat stará data a spolu s tím i zálohovat. Musíte tedy provádět čištění záloh zvlášť, třeba takto:

```
rdiff-backup /home/data /mnt/extdisk
rdiff-backup --remove-older-than 30D /mnt/zaloha
```

Kromě relativního nebo absolutního času je možné specifikovat i počet záloh, které se mají nechat, tzn. pokud chcete nechat pouze deset posledních záloh a ostatní smazat, použijete následující příkaz:

```
rdiff-backup --remove-older-than 10B /mnt/zaloha
```

## Správa linuxového serveru: Zálohování pomocí Duplicity

V minulém díle jsem vám představil zálohovací nástroj rdiff-backup. Zajímavou alternativou k tomuto nástroji představuje program Duplicity, který představím v tomto díle.

### Úvod

Na rozdíl od rdiff-backup není u Duplicity dostupná poslední záloha v „otevřené" podobě, kde by stačilo prosté překopírování souborů z úložiště zpět. Jiný způsob práce umožňuje tomuto nástroji nabídnout poněkud jiné spektrum funkcí jako například komprimaci či šifrování záloh pomocí GnuPG, nebo třeba v případě vzdáleného zálohování absenci nutnosti mít Duplicity i na cílovém počítači. Stejně jako rdiff-backup využívá i Duplicity přenosové pásmo efektivně a přenáší na vzdálený počítač pouze změny. Duplicity však může stejně jako rdiff-backup pracovat lokálně, tedy zálohovat v rámci jednoho počítače, a zvládá jak plnou, tak rozdílovou zálohu.

### Jednoduchá nešifrovaná záloha

Duplicity ve výchozím nastavení rovnou šifruje, tudíž pokud chcete jen jednoduchou, nešifrovanou zálohu, musíte šifrování explicitně zakázat. Cíl zálohy je nutné specifikovat pomocí URL včetně použitého protokolu – pro místní zálohu se používá protokol file:

```
duplicity --no-encryption /home/co file:///backup/kam
```

Duplicity prozkoumá cíl a sám zjistí, jestli má provést plnou nebo rozdílovou zálohu. Po dokončení práce vypíše Duplicity statistiku o proběhlé záloze, která vypadá takto:

```
--------------[ Backup Statistics ]--------------
StartTime 1281416525.29 (Tue Aug 10 07:02:05 2010)
EndTime 1281416528.38 (Tue Aug 10 07:02:08 2010)
ElapsedTime 3.08 (3.08 seconds)
SourceFiles 1617
SourceFileSize 694597580 (662 MB)
NewFiles 0
NewFileSize 0 (0 bytes)
DeletedFiles 0
ChangedFiles 0
ChangedFileSize 0 (0 bytes)
ChangedDeltaSize 0 (0 bytes)
DeltaEntries 0
RawDeltaSize 0 (0 bytes)
TotalDestinationSizeChange 3438 (3.36 KB)
Errors 0
-------------------------------------------------
```

### Vzdálené zálohování a šifrování záloh

Duplicity umožňuje zálohovat vzdáleně pomocí celé řady protokolů, počínaje klasickým unixovým ssh/scp přes rsync, ftp, WebDAV až po Amazon S3. Vzdálené zálohování prostřednictvím protokolu SCP vypadá takto:

```
duplicity --no-encryption /home/co scp://uzivatel@server/backup/kam
```

U vzdáleného zálohování je však vhodné šifrovat. Díky tomu je možné bez obav zálohovat na servery, které nemáte pod kontrolou, popřípadě na jiné své servery, kde díky šifrování víte, že i kdyby došlo k jejich kompromitaci, vlastní zálohy nebudou případnému útočníkovi bez znalosti klíče k ničemu. Zálohování se šifrováním je jednodušší než zálohování bez šifrování, stačí jej nezakazovat:

```
duplicity /home/co scp://uzivatel@server/backup/kam
```

Pokud si vyzkoušíte výše uvedený příkaz, zjistíte, že po vás Duplicity bude chtít heslo – implicitně se totiž k šifrování používá symetrická šifra s klíčem v podobě obyčejného hesla. Pro běžné použití je to asi postačující, ale pro pravidelné automatické zálohování (např. pomocí cronu) to není to pravé ořechové. Pokud vám stačí symetrická šifra, ale nechcete heslo pokaždé zadávat, můžete heslo umístit do proměnné prostředí PASSPHRASE takto:

```
PASSPHRASE="moje_dlouhe_tajne_a_ne_moc_bezpecne_heslo"
duplicity /home/co scp://uzivatel@server/backup/kam
```

#### *Šifrování pomocí PGP/GPG klíče*

Spíše než symetrickou šifru s heslem bývá lepší použít PGP/GPG klíč. Jeho ID můžete specifikovat jako parametr volbě --encrypt-key a použít jej místo obyčejného hesla:

```
duplicity --encrypt-key CAC193E6 /home/co scp://uzivatel@server/backup/kam
```

Jedna věc, která možná není úplně patrná a může se vám za jistých okolností hodit, je skutečnost, že klíčů můžete použít více najednou (poté můžete zálohu dešifrovat kterýmkoliv z nich):

```
duplicity --encrypt-key CAC193E6 --encrypt-key ACC91E63 /home/co scp://uzivatel@server/backup/kam
```

Ačkoliv při prvním provedení výše uvedených příkazů nebude Duplicity vyžadovat žádné heslo, při pokusu vytvořit rozdílovou zálohu opakováním stejného příkazu se vás Duplicity začne ptát na heslo k soukromému klíči. Je to dáno tím, že šifrováno je opravdu všechno, včetně seznamu zálohovaných souborů, který ovšem v rámci rozdílové zálohy Duplicity potřebuje, aby zjistil rozdíly od poslední zálohy. Pokud nechcete heslo zadávat ručně, můžete použít proměnnou prostředí PASSPHRASE, jak je naznačeno výše:

```
PASSPHRASE="moje_dlouhe_tajne_a_ne_moc_bezpecne_heslo"
duplicity --encrypt-key CAC193E6 /home/co scp://uzivatel@server/backup/kam
```

### Vyloučení položek z adresáře k zálohování

V řadě situací budete chtít zálohovat z příslušných adresářů pouze určité soubory, ale ne úplně všechny (některé soubory či adresáře je vhodné vyloučit už z principu jako /dev, /proc, /sys, atd.). Za tímto účelem obsahuje Duplicity volbu --exclude, kterou můžete použít opakovaně k vyloučení libovolného množství položek:

```
duplicity --exclude /home/co/vyradit --exclude /home/co/dev /home/co scp://uzivatel@server/backup/kam
```

V příkladu výše došlo k vyřazení adresářů /home/co/vyradit a /home/co/dev. Existuje však i volba --include, která slouží k opaku – začlenění adresáře či souboru pro zálohování, i když je jinde specifikován k vyřazení (pokud je např. vyřazen nadřazený adresář). Tuto možnost demonstruje následující příkaz, kde dojde k zálohování adresářů /home a /etc z kořenového adresáře:

```
duplicity --include /home --include /etc --exclude '**' / scp://uzivatel@server/backup/kam
```

### Obnova dat

Stejně jako v případě rdiff-backup postačí i v případě Duplicity přehodit pořadí adresářů a dojde k obnově dat z poslední úspěšně provedené zálohy:

```
duplicity --encrypt-key CAC193E6 scp://uzivatel@server/zaloha/ /home/obnoveno
```

Všimněte si, že je třeba explicitně specifikovat klíč k obnově dat. Pokud byste ho neuvedli, Duplicity by předpokládal použití symetrického klíče a vyzval vás k zadání symetrického hesla. Přirozeně, obnova probíhá pomocí soukromého klíče, ke kterému budete muset heslo zadat.
Tento příklad obnovil aktuální data k datu poslední zálohy, což ovšem ne vždy využijete. Někdy budete potřebovat obnovit data k některému dřívějšímu datu. To je možné provést taktéž, a sice pomocí volby -t a jejího parametru v podobě časového údaje. Čas můžete specifikovat buď absolutně (např. 2010-09-15), nebo relativně (např. 30D k obnově zálohy staré třicet dní). Obnova týden staré zálohy by vypadala takto:

```
duplicity --encrypt-key CAC193E6 -t 7D scp://uzivatel@server/zaloha/ /home/obnoveno
```

Výše uvedené příklady obnoví veškerá data, tzn. všechny zálohované soubory a adresáře. Pokud však chcete obnovit jediný soubor nebo adresář, toto chování se vám nebude zamlouvat a budete hledat způsob, jak obnovit pouze to, co potřebujete. K tomu slouží volba --file-to-restore:

```
duplicity --file-to-restore etc/apache2 -t 7D scp://uzivatel@server/zaloha/ /home/obnoveno
```

Výše uvedený příkaz obnoví sedm dní starý obsah adresáře etc/apache2 do /home/obnoveno.

### Plná záloha na povel a údržba úložiště se zálohami

Duplicity sám volí typ zálohy analýzou obsahu cílového úložiště. Pokud je úložiště prázdné, provede se plná záloha, v opačném případě se provede rozdílová. Jednou za určitou dobu je však vhodné provést plnou zálohu a staré zálohy pak smazat. Abyste provedli plnou zálohu do existujícího úložiště, použijte parametrfull:

```
duplicity --encrypt-key CAC193E6 full /home/co scp://uzivatel@server/backup/kam
```

Tím zajistíte, že se provede plná záloha bez ohledu na stav úložiště. Pokud víte, že chcete provést plnou zálohu vždy jednou za určitý časový interval, nemusíte postupovat výše naznačeným způsobem, můžete využít volby --full-if-older-than s parametrem v podobě určení času a nechat Duplicity, ať sám rozhodne, kdy provést plnou zálohu:

```
duplicity --encrypt-key CAC193E6 --full-if-older-than 7D /home/co scp://uzivatel@server/backup/kam
```

Výše uvedený příkaz provede plnou zálohu, pokud byla poslední plná záloha provedena před více než týdnem. Je jasné, že všechny rozdílové zálohy se váží k poslední provedené plné záloze. Chcete-li úložiště pro zálohování očistit od starých záloh, můžete odmazat některou z dřívějších plných záloh spolu se všemi rozdílovými, které na ní závisí. Abyste to mohli udělat, musíte nejprve vytvořit více plných záloh způsobem naznačeným výše. Jakmile budete mít více plných záloh s určitým časovým odstupem, můžete dřívější zálohy vymazat pomocí voleb remove-older-than či remove-all-but-n-full. První z uvedených voleb smaže všechny zálohy starší udaného času (samozřejmě nedojde k smazání žádné plné zálohy, na které závisí rozdílové zálohy provedené po specifikované době):

```
duplicity remove-older-than 30D scp://uzivatel@server/backup/kam
```

Výše uvedený příkaz smaže plné zálohy starší než třicet dní spolu s na ně navázanými rozdílovými, pokud ovšem tyto rozdílové zálohy nebyly pořízeny v průběhu posledních třiceti dní. Pokud jste tedy vytvořili poslední plnou zálohu před 31 dny, smažou se plné zálohy provedené před 31 dny, ale už ne ta, na které závisí rozdílové zálohy mladší třiceti dní.

Volba remove-all-but-n-full pracuje malinko jinak - ta bere jako parametr číslo odpovídající počtu plných záloh (s navázanými rozdílovými), které chcete zachovat. Pokud tedy zadáte jako parametr pro tuto volbu číslo tři, budou ponechány tři poslední plné zálohy (samozřejmě s na ně navázanými rozdílovými), ostatní budou vymazány.

## Správa linuxového serveru: Instalace LAMP

LAMP je zkratkou pro Linux, Apache, MySQL a PHP. Tento díl seriálu se bude zabývat základním zprovozněním Apache, MySQL a PHP v distribuci Debian. Nainstalována bude i webová aplikace phpMyAdmin sloužící k pohodlné správě databáze.

### Úvod

Na úvod je třeba zdůraznit, že pro jednotlivé komponenty ze zkratky LAMP existují alternativy, které mohou být v mnoha situacích příhodnější. Už samotný webový server Apache má v prostředí GNU/Linuxu poměrně velkou konkurenci. Existují odlehčené varianty snažící se o vyšší výkon jako např. Lighttpd, na kterém běží tak gigantické projekty jako Youtube, Wikipedia atd., ale také třeba Nginx. Zajímavým projektem hodným zmínky je i webový server Cherokee, který poskytuje přehledné webové konfigurační rozhraní, čímž odstraňuje nutnost editovat konfigurační soubory ručně, a ještě k tomu je nenáročný na zdroje a nabízí takévelmi slušný výkon.

Pokud se posunete k dalšímu písmenku, tedy k databázovému systému MySQL, i zde je v prostředí GNU/Linuxu značná konkurence. Na mysl by vám zde určitě měl přijít PostgreSQL, Firebird a další. Pokud vás zajímají rozdíly jednotlivých databázových systémů, můžete si přečíst článek Srovnání databázových serverů, nebo se podívat na komplexní srovnání databázových systémů na Wikipedii.

Posledním písmenkem ve zkratce LAMP je písmeno P, značící PHP. Ačkoliv se jedná o jeden z nejpoužívanějších jazyků pro tvorbu webových aplikací, má svoje nedostatky (i když řadu z nich se snaží řešit řada PHP frameworků) a existuje opět řada alternativ, z nichž budu jmenovat alespoň Javu, Python či Ruby.

### *Instalace Apache*

Samotná instalace Apache je v Debianu, ale i v řadě jiných distribucí velmi triviální – postačí nainstalovat jediný balíček – v případě Debianu balíček apache2:

```
aptitude install apache2
```

Po instalaci dojde ke spuštění serveru a vy se můžete na web rovnou podívat (postačí do prohlížeče zadat IP adresu serveru). Měl by se objevit známý nápis „*It works!*" značící, že je všechno v pořádku a webový server běží. Pokud pracujete vzdáleně přes SSH a máte k dispozici některý z konzolových webových prohlížečů (např. lynx), můžete webový server otestovat lokálně tímto způsobem:

```
lynx localhost
```

Apache očekává webovou prezentaci v adresáři /var/www, a to je také umístění, ze kterého v tomto případě zobrazil obsah souboru index.html.

### *Instalace PHP*

Pokud se jedná o základní instalaci PHP, ani zde není příliš velký problém – je ovšem potřeba nějakým způsobem PHP s Apachem propojit. O to se postará balíček libapache2-mod-php5:

```
aptitude install libapache2-mod-php5 php5
```

Po provedení tohoto příkazu restartujte Apache:

```
/etc/init.d/apache2 restart
```

V tuto chvíli by bylo dobré otestovat, zda vše korektně funguje a PHP aplikace vám poběží. Proto vytvořte soubor `/var/www/index.php` s následujícím obsahem:

```
<?php phpinfo(); ?>
```

Jelikož HTML soubory dostanou před PHP soubory přednost, můžete buď index.html ve /var/www vymazat, nebo zadat do prohlížeče explicitně cestu k index.php. Poté byste měli vidět uvítací stránku PHP detailně popisující konfiguraci. Pokud máte na serveru k dispozici Lynx, můžete se přesvědčit pomocí něj:

```
lynx localhost/index.php
```

Pokud dostanete místo popisu konfigurace PHP obsah souboru index.php v textové podobě, máte problém s konfigurací. Buď jste nerestartovali Apache, nebo jste nenainstalovali balíček libapache2-mod-php5.

Proběhlo-li vše v pořádku, máte k dispozici webový server schopný běhu PHP aplikací. PHP je ovšem modulární, takže v tuto chvíli máte k dispozici pouze základní PHP bez řady doplňků, které webové aplikace často využívají. Bývá dobrou praxí instalovat pouze ty doplňky, které opravdu potřebujete. Podívejte se tedy do dokumentace aplikací, které chcete na serveru provozovat, a nainstalujte doplňky pro PHP dle toho. Seznam doplňků, které jsou dostupné v Debianu, získáte kupříkladu takto:

```
aptitude search php5
```

### *Instalace MySQL*

Instalace MySQL obnáší nutnost nainstalovat PHP modul, který je schopný s MySQL pracovat, konkrétněphp5-mysql. Samotnou instalaci MySQL serveru, klienta a modulu pro PHP zajistíte následujícím příkazem:

```
aptitude install php5-mysql mysql-server
```

Během instalace budete požádáni o zadání hesla pro administrátorský účet databáze. Důrazně se doporučuje této možnosti využít a administrátorský účet heslem ochránit. To platí dvojnásob v situaci, kdy zpřístupníte rozhraní databáze přes webové stránky veřejnosti, pomocí aplikace phpmyadmin.

Pokud využíváte řádkového klienta MySQL a vadí vám nutnost zadávat heslo, můžete si pomoci vytvořením využít konfigurační soubor MySQL klienta $HOME/.my.cnf, kam můžete heslo uvést, aby se vás na něj řádkový MySQL klient nemusel ptát. Obsah daného souboru upravte podle následujícího vzoru:

```
[client]
password=heslo
```

### *Instalace aplikace phpMyAdmin*

PhpMyAdmin je webová aplikace napsaná v PHP a určená k pohodlné správě MySQL databáze. Představuje možnost, jak pracovat s databázovým systémem rychle, přehledně a efektivně bez nutnosti psát všechny SQL příkazy ručně. V případě LAMP serverů je této aplikace hojně využíváno také kvůli možnosti snadného importu a exportu dat z nebo do databáze. Více informací o použití této aplikace naleznete v článku Správa databází pomocí phpMyAdmin. Instalaci nástroje phpMyAdmin provedete v Debianu takto:

```
aptitude install phpmyadmin
```

V průběhu instalace se vás instalátor dotáže, zdali má zkonfigurovat některé servery pro běh phpmyadmina – zde zvolte apache2. Díky tomu budete moci po instalaci začít aplikaci phpMyAdmin rovnou používat – bude dostupná v umístění /phpmyadmin relativně ke kořenu serveru:

```
lynx localhost/phpmyadmin
```

Budete-li k této aplikaci přistupovat vzdáleně, čiňte tak raději přes HTTPS, až tímto způsobem webový server nastavíte. Budete-li používat nešifrovaný protokol HTTP, umožníte kterémukoliv naslouchajícímu na cestě mezi vámi a vaším serverem, aby vaše heslo k databázi odposlechl.

### *LAMP je (skoro) hotov*

V tuto chvíli již máte nainstalován funkční webový server s databází MySQL a podporou pro PHP aplikace. Můžete si vyzkoušet práci s databází prostřednictvím phpMyAdmina, popřípadě si na server zkusit nahrát nějakou PHP aplikaci a zkusit ji zprovoznit. Jistě jste si všimli, že nebylo třeba zasahovat do konfiguračních souborů – bylo využito výchozí konfigurace Debianu. Bohužel tato výchozí konfigurace má své nedostatky. Předně PHP je nastaveno v režimu navrženém pro vývojové, nikoliv pro produkční servery (nevyužívá tedy adekvátní možnosti zabezpečení pro produkční servery). Dále Apache v tuto chvíli nabízí jedinou prezentaci ve /var/www pro všechny domény, které jsou na webový server nasměrované. Není také nakonfigurované SSL, takže není přístupná varianta webu přes zabezpečený protokol HTTPS. Konfiguraci a možnostem zabezpečení se budou věnovat další díly.

## Správa linuxového serveru: Konfigurace LAMP: PHP

V předchozím díle byla probrána instalace platformy LAMP (zkratka pro Linux, Apache, MySQL a PHP). Tento díl seriálu se bude věnovat základům konfigurace LAMP v prostředí Debianu, konkrétněji konfiguraci a možnostem zabezpečení PHP interpretu.

### Úvod

Hlavním konfiguračním souborem PHP je php.ini, sídlící v adresáři /etc/php5/apache2. Všimněte si, že tento konkrétní konfigurační soubor je umístěn v podadresáři apache2. PHP interpret může mít odlišný konfigurační soubor pro různé typy použití, které jsou pak umístěny v podadresářích /etc/php5. Kupříkladu, pokud nainstalujete balíček php5-cli, který obsahuje interpret PHP pro příkazovou řádku, dojde k vytvoření konfiguračního souboru php.ini v adresáři /etc/php5/cli, který se použije v rámci tohoto interpretu. Jeho konfigurace tak bude oddělena od konfigurace PHP určené pro webový server Apache v umístění uvedeném výše.

PHP má řadu rozšíření (extensions), které jsou v Debianu k dispozici v podobě balíčků s prefixem php5-, popřípadě php-. Naleznete zde konektory k databázím (např. php5-mysql, php5-pgsql či php5-sqlite), rozšíření pro práci s obrázky (php5-gd a php5-imagick) a řadu dalších. Různé webové aplikace vyžadují nebo umí využít různá rozšíření, o čemž se obvykle dozvíte v jejich dokumentaci. Vhodnou strategií bývá instalovat rozšíření teprve, až když víte, že jej budete opravdu potřebovat.

### *php.ini*

Konfigurační soubor php.ini je (nejen) v Debianu velmi dobře komentovaný, je tedy vhodné doporučit, abyste si jej celý prošli a nastavili podle svých požadavků. To platí dvojnásob, pokud konfigurujete produkční server. V souvislosti s tím vás raději znovu upozorním na to, že výchozí php.ini určený pro Apache je v Debianu nastavený spíše s ohledem na vývojové servery, nikoliv na produkční. Na produkčních serverech je vhodné konfiguraci upravit přinejmenším s ohledem na bezpečnost a na požadavky provozovaných aplikací. Debian Lenny nabízí php.ini určený pro produkční použití v/usr/share/doc/php5/examples/php.ini-recommended.

Samotná rozšíření pak naleznete jako samostatné .ini soubory v /etc/php5/conf.d, kde v některých případech naleznete kromě direktivy pro zavedení příslušného modulu i specifické konfigurační volby. V jednotlivých podadresářích s php.ini (např. /etc/php5/apache2) se pak nachází symbolický odkaz conf.dvedoucí na hlavní /etc/php5/conf.d, tzn. zavedení modulů je pro jednotlivé konfigurace společné, ale není problém symbolický odkaz nahradit adresářem, kde můžete sami rozhodnout, které moduly mají být v dané konfiguraci použity.

Mezi důležité konfigurační volby, kterým byste měli věnovat pozornost, patří následující:

expose_php přijímá jako parametr „On" nebo „Off" a umožňuje „skrýt" přítomnost PHP v HTTP hlavičkách odpovědí webového serveru. Pokud je nastavena na „On", ohlašuje Apache přítomnost PHP i verzi takto:

```
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch
```

Je-li volba expose_php nastavena na „Off", přítomnost PHP není ohlašována vůbec:

```
Server: Apache/2.2.9 (Debian)
```

max_execution_time přijímá jako parametr čas v sekundách a určuje, jak dlouho může PHP skript běžet před tím, než bude násilně ukončen. Výchozí hodnotou je 30 (vteřin).

memory_limit přijímá jako parametr maximum paměti, kterou může PHP skript využít. Výchozí hodnotou je128M, kde M znamená MB. Pokud skript požádá o více paměti, bude ukončen s chybou. Jak v případě volbymax_execution_time, tak v případě této platí daná hodnota pro jeden běh skriptu, tedy jeden HTTP požadavek obsluhovaný PHP skriptem. Více požadavků naráz může tedy zabrat n-krát více paměti.

display_errors řídí zobrazování chyb a může nabývat dvou hodnot, „On" nebo „Off". Jeho výchozí hodnota je „On", což představuje potenciální bezpečnostní problém - chyby, varování či hlášení v rámci běhu PHP skriptů se pak zobrazují v prohlížeči včetně některých možných zneužitelných informací (chybová hlášení při práci s databází atd.). Tuto volbu je tedy na produkčních serverech vhodné nastavit na „Off". To, co se vlastně zobrazuje, tzn. zdali se zobrazují pouze chyby, nebo i varování a hlášení, řídí parametrerror_reporting. Více k tomuto tématu naleznete např. v tomto článku.

display_startup_errors řídí zobrazování kritických chyb, ke kterým dochází při spouštění PHP skriptu. Nabývá stejných hodnot jako parametr display_errors, jeho výchozí hodnotou je „Off" a na produkčních serverech doporučuji tuto hodnotu ponechat.

log_errors vám pomůže se k chybovým hlášením dostat, pokud vypnete jejich zobrazování. Nabývá hodnot „On" nebo „Off", přičemž hodnota „On" zařídí logování chybových hlášení do chybového logu webového serveru. Pokud budete logování chybových hlášení povolovat, podívejte se i na další parametry ovlivňující logování, např. parametr ignore_repeated_errors, kterému nastavte hodnotu „On", aby se zbytečně neplnil váš log opakovanými chybovými hláškami, či samotný výše zmíněný parametr error_reporting ovlivňující typy hlášení, na které se bude brát zřetel.

disable_functions umožňuje zakázat provádění jistých PHP funkcí ve skriptech, což může posloužit jako jedno z bezpečnostních opatření. Příkladem seznamu funkcí k zakázání může být tento:

```
disable_functions = shell_exec, exec, system, passthru, proc_close, proc_open, proc_get_status, proc_nice, proc_open, proc_terminate, shell_exec, dl, popen, pclose, chown, disk_free_space, disk_total_space, diskfreespace, fileinode, max_execution_time, set_time_limit, highlight_file, show_source, phpinfo, chgrp, symlink
```

Pokud budete tento parametr používat, dejte si pozor na funkce s obdobnou funkcionalitou, ale jiným názvem. Stejně tak doporučuji bezpečnostní dopad jednotlivých funkcí zvážit - v seznamu výše může být považována za diskutabilní např. funkce phpinfo, která „pouze" umožňuje zobrazit podrobnou konfiguraci PHP.

allow_url_fopen a allow_url_include jsou vhodnými kandidáty na hodnotu „Off". Umožňují v rámci některých funkcí určených pro práci se soubory získávat data z externích zdrojů (specifikováním URL místo cesty k souboru), což může posloužit k nejrůznějším útokům, pokud není v PHP aplikacích správně ošetřen uživatelský vstup (na což nebývá dobré se spoléhat).

open_basedir je Debianem označeno jako „defektní" (broken) bezpečnostní opatření, a je tudíž v rámci Debianu nepodporované. open_basedir slouží k omezení operací se soubory na seznam adresářů uvedených jako hodnota této volby (adresáře je třeba oddělovat dvojtečkou). Pokud je tato volba ponechána jako neprázdná, nebude možné v rámci korektně ošetřených PHP funkcí použít cestu vedoucí mimo tyto adresáře (resp. použití takové cesty vyvolá chybu). Lepším (resp. jistějším) způsobem zabezpečení může být chroot na úrovni webového serveru (Apache má k tomuto účelu připraven modulchroot v balíčku libapache2-mod-chroot).

Na závěr výčtu důležitých voleb dodávám, že některé z výše uvedených navrhovaných změn mohou mít negativní dopad na některé PHP aplikace. Dodám také, že veškeré změny v php.ini se projeví teprve až se znovunačtením konfigurace webového serveru, tedy v případě Apache po vykonání příkazu:

```
/etc/init.d/apache2 reload
```

Staré a nepodporované volby

Ve výčtu důležitých konfiguračních voleb jsem vynechal dvě, které jsou obecně považovány za zastaralé (deprecated), nepodporované či dokonce za nebezpečné. Jejich výchozí hodnoty jsou obvykle nastaveny na „Off" a bývá doporučeno je tak ponechat. Jedná se o volby register_globals a safe_mode. První z uvedených je dobře známá a historicky velmi kontroverzní funkce stojící za řadou bezpečnostních zranitelností, zatímco ta druhá je pokusem o zvýšení bezpečnosti na sdíleném hostingu, který se neujal, má některé problémy. Řada webových aplikací jej nemá ráda a je momentálně považován za zastaralý (deprecated) a nepodporovaný.

### *Dodatečné zabezpečení PHP: suhosin a suexec*

Mezi dvě dodatečné možnosti zabezpečení PHP patří projekt Suhosin a nástroj (resp. modul) suexec. Projekt Suhosin se sestává jednak z bezpečnostního patche jádra PHP, který se snaží PHP interpret obrnit vůči některým typům útoků, a jednak z rozšíření PHP, které nabízí další možnosti zabezpečení. Interpret PHP v Debianu v sobě již Suhosin patch integruje, ale samotné rozšíření ve výchozí instalaci není. Toto rozšíření je k dispozici v balíku php5-suhosin. Po jeho instalaci budete mít k dispozici jeho konfiguraci v/etc/php5/conf.d/suhosin.ini. Samotnou konfigurací Suhosinu se tento díl zabývat nebude - pokud o toto téma máte zájem, podívejte se na odkazy v závěru článku.

Suexec je modul Apache určený k tomu, aby PHP skript běžel s oprávněními vlastníka daného webu. Tím je řešen bezpečnostní problém v konfiguraci s řadou webů (virtualhostů), kde průnik přes jeden z nich umožní útočníkovi dostat se i k ostatním. Suexec vyžaduje běh PHP v rámci Apache prostřednictvím CGI, tedy přesněji FastCGI. Toto řešení je popsáno třeba v tomto článku.

## Správa linuxového serveru: Úvod do konfigurace Apache

V tomto dílu bude probrán základ konfigurace Apache v prostředí Debianu, který má ke konfiguraci Apache specifický přístup. V druhé části článku budou probrány virtuální weby (virtual hosts), tedy možnosti provozu více různých webových prezentací na jednom webovém serveru.

### Úvod

Jelikož konfigurace Apache má bohaté možnosti a je velmi komplexní, mají různé distribuce různý přístup k jeho konfiguraci. V Debianu je konfigurace Apache rozdělena do modulární struktury, která se nachází v/etc/apache2. Výhodou takto rozdělené konfigurace je relativně jednoduchá obsluha i velmi komplikovaného nastavení, neb místo hledání v jednom sáhodlouhém konfiguračním souboru sáhnete po mnohem menším, který obsahuje nastavení relevantní pro konkrétní „virtual host" nebo modul.

Hlavním konfiguračním souborem je apache2.conf. Pokud budete potřebovat upravit nastavení uložená v tomto souboru, popřípadě budete chtít nastavení webového serveru něčím doplnit, je lepší za tímto účelem použít httpd.conf, který je k tomu určený. Jednak tím získáte přehled o změnách, které jste provedli vůči distribuční konfiguraci, a jednak se vyhnete problémům při aktualizaci, kdy se může distribuční konfigurace změnit, a vy pak budete muset rozdíly v distribuční konfiguraci zpracovat a zahrnout ručně.

Ke konfiguraci sítě, respektive IP adres a portů, kde bude Apache naslouchat, je určen soubor ports.conf(podrobnější popis viz níže). Následuje několik adresářů s řadou konfiguračních souborů a symbolických odkazů:

#### *conf.d*

Tento adresář je určen k umístění dodatečných konfiguračních souborů. Typicky zde naleznete různá doplňující nastavení, tématicky rozložená do jednotlivých souborů. Pokud máte nainstalovaný phpmyadmin z distribučních repozitářů, naleznete zde symbolický odkaz phpmyadmin.conf ukazující na konfigurační soubor, který upraví nastavení Apache tak, abyste mohli k webové aplikaci phpmyadmin přistupovat prostřednictvím aliasu /phpmyadmin. Repozitáře Debianu obsahují řadu webových aplikací, z nichž mnohé jsou konfigurovány pro Apache tímto způsobem. Pokud byste si nainstalovali třeba Drupal (balíček drupal6 v Debianu Lenny), naleznete tu také příslušný symbolický odkaz vedoucí ke konfiguračnímu souboru s nastavením Apache pro Drupal.

#### *mods-available a mods-enabled*

Pomocí modulů můžete měnit funkčnost webového serveru, respektive přidávat a odebírat různé vlastnosti. Součástí distribuce Apache je sada základních, nejčastěji používaných modulů. V repozitářích Debianu, ale i mnoha jiných distribucí, naleznete řadu doplňkových modulů. Pokud jste se drželi návodu na instalaci LAMPo dva díly zpět, nainstalovali jste jeden z doplňkových modulů v balíčku libapache2-mod-php5, který zprostředkovává použití PHP interpretu ve webových stránkách.

Moduly je třeba zavést, popřípadě ještě nastavit. Konfigurační soubory pro dostupné moduly naleznete v adresáři mods-available (hledejte příslušný soubor s příponou .conf). Soubory s příponou .load zajišťují pouze natažení modulu.

To, co je v adresáři mods-available, však Apache při svém spuštění ignoruje - bere v úvahu pouze to, co je k dispozici v adresáři mods-enabled. Tento adresář obsahuje obvykle pouze symbolické odkazy vedoucí ke konfiguračním souborům modulů v mods-available, které se mají zavést. Díky tomuto mechanismu se natáhnou pouze ty moduly, které chcete. Abyste nemuseli příslušné symbolické odkazy vytvářet ručně, má Debian k dispozici dva nástroje na práci s moduly - a2enmod („zapíná" moduly) a a2dismod („vypíná" moduly). Oba nástroje lze použít jak v příkazovém režimu, kde parametr tvoří jméno modulu, který má být aktivován či deaktivován, tak v interaktivním režimu (k tomu postačí daný nástroj spustit bez parametru).

#### *sites-available a sites-enabled*

Podobný princip konfigurace a správy připravili správci Debianu i pro práci s virtuálními weby (virtual host). Jejich konfigurační soubory jsou umístěny v sites-available, přičemž Apache tento adresář v úvahu nebere, naopak bere v úvahu vše, co je v adresáři sites-enabled. Jednotlivé virtuální weby lze tedy snadno „vypínat" a „zapínat", podobně jako moduly, a to buď ručním vytvořením příslušného symbolického odkazu, nebo použitím nástrojů a2ensite a a2dissite. S těmito nástroji se pracuje úplně stejně jako s těmi určenými pro práci s moduly.

### Základy konfigurace Apache

Po nainstalování Apache jsou aktivní některé základní moduly a jeden virtuální web (sites-available/default). Co je vlastně virtuální web (virtual host)? Virtuální web je koncept, který slouží k provozu více webů (či domén) na jednom webovém serveru. Je možné provozovat různé weby na různých IP adresách či mnoho domén na jedné nebo více IP adresách (popř. ještě i různých portech), a to vše v rámci jednoho běžícího webového serveru. Jednotlivé weby či domény je však třeba od sebe oddělit, aby Apache poznal, který HTTP požadavek má směrovat na který virtuální web. A právě k tomu slouží v případě Apache direktiva VirtualHost (více viz níže).

#### *Konfigurace sítě*

Základem práce webového serveru je práce se sítí. Naslouchání na určité IP adrese a portu se nastavuje prostřednictvím volby Listen v souboru ports.conf, tedy např:

```
Listen 80
```

Tato volba zapříčiní, že Apache bude poslouchat na portu 80 všech IP adres, které má server k dispozici. Pokud chcete omezit Apache na jednotlivé IP adresy či porty, můžete to provést takto:

```
Listen 1.2.3.4:80
Listen [2001:db8:a1:b2:c3:d4:e5:f6]:80
```

Z příkladu výše je patrné jednak to, že direktiv Listen je možno použít více, a jednak to, že IPv6 adresy je třeba oddělit hranatými závorkami.

Pár slov o virtuálních webech a HTTPS

Virtuální weby mohou být od sebe odděleny dvěma způsoby, buď na základě jména (name-based), nebo na základě IP adresy (IP-based). Můžete tedy mít více různých webů na různých IP adresách, nebo jednu či více domén směřující na jednu nebo více IP adres. Problém nastane v případě protokolu HTTPS, kde jmenné rozlišování z principu nefunguje. Je tomu tak proto, že se nejprve vytváří SSL spojení, a teprve pak se v rámci již vytvořeného SSL spojení přenáší HTTP požadavek obsahující doménové jméno. To znamená, že webový server nemůže zjistit příslušnost k virtuálnímu webu podle jména. Díky tomu je na jedné IP adrese možné provozovat pouze jediný HTTPS virtuální web. Na jednom HTTPS virtuálním webu sice je možné jistým způsobem provozovat více domén (během dalších dílů bude osvětleno), ale SSL certifikát lze použít pouze jediný.

Aby Apache mohl rozlišovat virtuální weby podle jména, musí vědět, na kterých IP adresách a portech má podle jména vyhledávat. K tomu slouží direktiva NameVirtualHost, jejímž parametrem je IP adresa, popřípadě IP adresa doplněná číslem portu (oddělovačem je dvojtečka). Výchozí nastavení v Debianu naleznete v ports.conf v této podobě:

```
NameVirtualHost *:80
```

Toto nastavení zapříčiní, že na všech IP adresách dostupných webovému serveru bude aktivováno hledání virtuálních webů podle jména.

#### *Konfigurace virtuálních webů*

Definice virtuálního webu se řídí direktivou VirtualHost, která se používá úplně stejně jako HTML tag. Otevírací tag má parametr v podobě IP adresy a portu, kde bude virtuální web k dispozici. IP adres může být více (v takovém případě jdou za sebou, odděleny mezerou). Hvězdička na místě IP adresy značí, že daný virtuální web bude k dispozici na všech IP adresách, na kterých Apache naslouchá. Příkladem definice virtuálního webu může být toto:

```
<VirtualHost *:80>
ServerName www.domena1.tld
ServerAlias domena1.tld *.domena1.tld
DocumentRoot /var/www/domena1
</VirtualHost>
```

Uvnitř direktivy VirtualHost je dobré specifikovat příslušné doménové jméno v direktivě ServerName.ServerAlias obsahuje všechna alternativní doménová jména společná pro daný virtuální web. V tomto případě se ke stejnému webu dostanete, i pokud zadáte jen „domena1.tld" bez „www", nebo zadáte jinou doménu třetího řádu, např. „test.domena1.tld".

Umístění kořenu webové prezentace daného virtuálního webu je lze zadat jako parametr direktivyDocumentRoot. Pokud ji nezadáte, použije se výchozí, definovaná mimo všechny VirtualHost direktivy (čili např. v httpd.conf). Tato „dědičnost" funguje i v případě dalších direktiv. V případě, že není hodnota definovaná nikde, použije se hodnota výchozí.

V Debianu by definice virtuálního webu měla být umístěna do samostatného, vhodně pojmenovaného souboru v /etc/apache2/sites-available. Daný virtuální web pak povolíte nástrojem a2ensite a restartem Apache uvedete do provozu.

Další virtuální web byste pak přidali vytvořením dalšího souboru v sites-available s obdobným obsahem, např. takto:

```
<VirtualHost *:80>
```

```
ServerName www.domena2.tld
ServerAlias domena2.tld *.domena2.tld
DocumentRoot /var/www/domena2
</VirtualHost>
```

Poté byste použili obdobný postup pro „aktivaci" daného webu (nástroj a2ensite). Tímto postupem byste získali dva virtuální weby, jeden s doménou domena1.tld a vlastním obsahem webu dostupným ve/var/www/domena1, druhý s doménou domena2.tld a obsahem webu v /var/www/domena2. Oba by mohly běžet v rámci jedné instance Apache třeba na jedné veřejné IP adrese.

***Plně kvalifikované doménové jméno***

Jeden z nejčastějších problémů řešeným v souvislosti s konfigurací Apache je varování, které říká, že Apache nemohl spolehlivě zjistit plně kvalifikované doménové jméno serveru. Tato hláška může vypadat třeba takto:

```
apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1 for ServerName
```

Aby mohl Apache správně fungovat, potřebuje o každém virtuálním webu znát dvě věci - jméno serveru a alespoň jednu jemu příslušnou IP adresu. Kromě toho ovšem Apache potřebuje znát IP adresu a doménové jméno samotného serveru. Toto hledá v globální definici ServerName (tj. definici mimo všechny direktivyVirtualHost, v Debianu nejspíše v /etc/apache2/httpd.conf). Pokud ji nenajde, pokusí se použít hostnamedaného serveru (v Debianu se hostname nastavuje v /etc/hostname). To se nejspíše povede, ale ne vždy je jako hostname použito plně kvalifikované doménové jméno (FQDN), a to je podstata problému. V řadě případů je jako hostname použito jen obyčejné slovo, popřípadě „localhost", a to FQDN není. V zásadě - FQDN musí obsahovat alespoň jednu tečku (např. example.org). Nejčastěji se za tímto účelem upravuje/etc/hosts, kde se přidává příslušný záznam, který může vypadat třeba takto:

```
1.2.3.4 example.org
```

Pokud ve jmenné části specifikujete více jmen, dejte si pozor, ať je FQDN na prvním místě, tj. následující příklad je špatně:

```
1.2.3.4 example example.org
```

Správně by bylo:

```
1.2.3.4 example.org example
```

**Pár slov na závěr**

Virtuální weby umožňují na jednom webovém serveru provozovat více různých webů rozlišených podle jména nebo IP adresy, nejčastěji pak podle doménového jména. Konkrétní postup nastavení a oddělení jednotlivých webových prezentací podle jména byl popsán v článku. Každý z virtuálních webů může mít specifikovány různé volby a možnosti, které jsou zapsané uvnitř direktivy VirtualHost. Některé z těchto voleb a možností, stejně jako některé z modulů webového serveru Apache, budou probrány v příštím díle.

## Správa linuxového serveru: Úvod do konfigurace Apache (pokračování)

V minulém díle byl probrán základ konfigurace virtuálních webů (virtual host). Tento díl je věnovaný konfiguraci v rámci jednotlivých virtuálních webů, včetně konfigurace přístupu k serveru prostřednictvím HTTPS.

### Úvod

Raději ještě jednou předesílám, že konfigurace Apache, která je zmíněna v tomto a v předchozím díle, je specifická pro distribuci Debian a distribuce na Debianu založené (např. Ubuntu). Jiné distribuce mohou mít odlišný mechanismus nastavování i umístění konfiguračních souborů. Mimo Debian nemusí být kupříkladu konfigurace Apache v /etc/apache2, ale např. v /etc/httpd (případ RHEL, CentOS, Fedory či Arch Linuxu). Podobně to pak bývá se spouštěcím skriptem, který bývá nazýván httpd, a nikoliv apache2. Mechanismy konfigurace a rozdělení konfiguračních souborů se také mohou lišit.

### Konfigurace v rámci virtuálních webů

Konfigurační možnosti Apache v rámci jednotlivých virtuálních webů jsou velice široké. Zde doporučuji podívat se do dokumentace, kde je vše dopodrobna popsáno. V tomto článku budou představeny pouze ty nejpoužívanější a nejzásadnější volby. Jednou z nejobvyklejších oblastí v rámci konfigurace virtuálních webů je řízení přístupu. To se obvykle specifikuje na úrovni adresářů, jejichž konfigurační volby se „obalí" do příslušného tagu, takto:

```
<Directory /var/www/example.com/>
Order deny,allow
Deny from 1.2.3.4
Allow from all
</Directory>
```

Zde je povolen přístup k webové prezentaci umístěné ve /var/www/example.com/ ze všech počítačů s výjimkou počítače 1.2.3.4. Na prvním řádku uvnitř konfigurace daného adresáře se nachází pořadí, v jakém bude Apache řídit přístup - pořadí deny,allow bude znamenat, že se nejprve projdou všechny volbyDeny, a pokud jim klient nevyhoví, přejde se na volby Allow. Pokud klient některé z voleb Deny vyhoví, bude mu odmítnut přístup (HTTP kód 403).

Na druhém řádku je specifikováno odmítnutí klienta s IP adresou 1.2.3.4. Na třetím řádku je pak povolen přístup odevšad. Je to právě pořadí provádění, které zajistí, že klientovi 1.2.3.4 bude odmítnut přístup. Kdyby bylo pořadí opačné, začalo by se nejprve u voleb Allow, kde by byly všechny přístupy povoleny a na příslušné Deny pravidlo by Apache nikdy nenatrefil. Na podobném principu lze velice jednoduše dosáhnout omezení přístupu k danému webu na pouze určité IP adresy, takto:

```
<Directory /var/www/example.com/>
Order allow,deny
Allow from 1.2.3.4
Allow from 4.3.2.1
Deny from all
</Directory>
```

V tomto případě je situace opačná, k webové prezentaci mohou přistupovat pouze počítače 1.2.3.4 a4.3.2.1, ostatním bude přístup zamezen.

#### *Options*

Důležitou konfigurační volbou pro konkrétní adresář je Options, která řídí, které vlastnosti serveru budou v daném adresáři k dispozici. Nejčastěji se zde vypínají funkce jako adresářový index, což je vlastnost umožňující jednoduše procházet adresářovou strukturu, pokud chybí adresářový index (typicky index.htmlči index.php, lze ovlivnit volbou DirectoryIndex). Vypnutí indexu provedete takto:

```
<Directory /var/www/example.com/>

Options -Indexes

Order allow,deny
Allow from 1.2.3.4
Allow from 4.3.2.1
Deny from all

</Directory>
```

V příkladu výše si povšimněte použití direktivy Options. Minus zde vypíná volbu Indexes. Ostatní možnosti v rámci Options se dočtete v dokumentaci.

#### *Aliasy*

Často používanou volbou jsou aliasy. Prostřednictvím aliasů můžete mít na určité URL navázaný jiný adresář či jinou webovou aplikaci. Pokud chcete mít například v rámci jednoho virtuálního webu aplikaci Squirrelmail navázanou na URL /mail, použijete alias, takto:

```
Alias /mail /usr/share/squirrelmail
<Directory /usr/share/squirrelmail>

Order allow,deny
Allow from all

<Files configtest.php>
order deny,allow
deny from all
allow from 127.0.0.1
</Files>

</Directory>
```

Dodávám, že tento příklad byl zkrácen, tzn. pokud budete instalovat na server Squirrelmail, použijte konfiguraci specifikovanou v dokumentaci (nebo výchozí konfiguraci Debianu), a nikoliv tento příklad. V tomto příkladu je vidět definice aliasu, kde URL /mail odkazuje na webovou aplikaci uloženou v/usr/share/squirrelmail, a také omezení přístupu k souboru configtest.php, ke kterému je z bezpečnostních důvodů povolen přístup pouze z místní smyčky (localhost).

#### *Konfigurace SSL/HTTPS*

Primárním mechanismem pro zprostředkování SSL spojení je v případě Apache modul mod_ssl, který je třeba nejprve aktivovat:

```
a2enmod ssl
/etc/init.d/apache2 restart
```

V Debianu je již k dispozici šablona pro virtuální web dostupný přes SSL, a sice /etc/apache2/sites-available/default-ssl, kterou můžete po úpravě aktivovat takto:

```
a2ensite default-ssl
/etc/init.d/apache2 restart
```

Klíčem k provozování SSL je však dostupnost SSL certifikátu. Zde máte na výběr ze dvou možností. Buď si vygenerujete „self-signed" (sám sebou podepsaný) SSL certifikát, na který vám dnešní prohlížeče zareagují velice odpudivě (viz obrázek níže) ve snaze odradit běžné uživatele od přístupu k takovému webu, nebo si necháte certifikát podepsat (či vygenerovat) od známé a prohlížeči uznávané certifikační autority. Problémem je, že takový certifikát není až na výjimky zdarma. Já osobně vím o dvou certifikačních autoritách, které vydávají certifikáty zdarma, a sice CAcert (stav podpory v prohlížečích) a StartSSL (stav podpory v prohlížečích). Pokud víte o nějaké další, určitě se o to podělte v komentářích pod článkem.


*Reakce prohlížeče Google Chrome na self-signed certifikát*

***Generování self-signed certifikátu***

Pokud máte nainstalovaný balíček ssl-cert, pak už máte nejspíše sebou podepsaný certifikát vygenerovaný a připravený k použití. V takovém případě nemusíte provádět vůbec nic, protože šablonadefault-ssl je na tento certifikát připravena. V opačném případě nainstalujte zmíněný balíček, popřípadě si vygenerujte vlastní certifikát s vlastními hodnotami pomocí openssl nebo make-ssl-cert.

***Použití vlastního certifikátu***

Máte-li k dispozici vlastní certifikát, vyplňte cestu k němu, k jeho klíči, popřípadě k dalším podstatným souborům, jak je naznačeno, do definice příslušného virtuálního webu (v tomto případě/etc/apache2/sites-available/default-ssl):

```
SSLCertificateFile    /cesta/k/certifikatu.pem
SSLCertificateKeyFile /cesta/ke/klici.key
```

***Omezení HTTPS a možnost jeho řešení pomocí SNI***

Jak už bylo zmíněno v minulém díle, certifikát je specifický už z podstaty HTTPS protokolu pro danou IP adresu. Pokud chcete použít více certifikátů pro různé weby, musíte nakonfigurovat příslušné virtuální weby podle IP adres a každému virtuálnímu webu přiřadit jinou IP adresu. Pokud si říkáte, že takové řešení je nešťastné, máte pravdu. Měli byste vědět, že existuje ještě jeden mechanismus pro vytváření HTTPS spojení, a sice SNI (Server Name Indication), kde je zmíněný problém odstraněn, a pomocí kterého je možné na jedné IP adrese provozovat více webů chráněných různými certifikáty. Návod pro zprovoznění tohoto řešení naleznete v odkazech pod článkem. Problémem SNI je nutnost podpory tohoto řešení u všech klientů, tzn. na straně prohlížečů, kde sice podpora je, ale pouze v novějších verzích. V Debianu Lenny podpora pro SNI v modulu mod_ssl chybí, je tedy nutno použít modul mod_gnutls, který je k dispozici v balíčku libapache2-mod-gnutls.

Tím bych tento díl ukončil. Příští díl se bude věnovat některým zajímavým modulům a jejich konfiguraci.

## Správa linuxového serveru: Úvod do konfigurace Apache (moduly rewrite a chroot)

V tomto dílu budou v rychlosti představeny dva moduly Apache, mod_rewrite sloužící k přepisu URL, který má široké možnosti použití počínaje „pěknými URL", a mod_chroot představující jistou formu zabezpečení formou izolace běhu Apache v části adresářové struktury.

### mod_rewrite aneb pěkná URL

Jedním z nejpoužívanějších modulů pro Apache je mod_rewrite, komplexní systém pro přepis či manipulaci s URL. Tento doplněk je využíván, v některých případech i přímo vyžadován řadou webových aplikací. I když jsou jeho možnosti velmi široké a je možné jej použít k mnoha účelům, nejčastěji je užíván pro tvorbu „pěkných" URL, tzn. skrývání standardních mechanismů pro předávání parametrů webové aplikaci tak, aby URL vypadalo „pěkně" a bylo přátelské pro roboty vyhledávačů (SEO). Pro představu, URL využívající standardního mechanismu pro předávání parametrů webové aplikaci vypadá třeba takto:

```
http://www.example.com/index.php?nazev_clanku=sprava_linuxoveho_serveru
```

Modul rewrite umožňuje příslušný parametr korektně předat webové aplikaci, přičemž však navenek využívá úplně jiné URL, např.:

```
http://www.example.com/clanky/sprava_linuxoveho_serveru.html
```

Jak to vlastně funguje? Pravidlo pro přepis umí „přeložit" výše uvedený příklad pěkného URL na původní, klasický tvar s parametrem pro index.php, kde může sídlit jádro webové aplikace. Webové aplikace se pak obvykle postarají o to, aby generovaly správná, „pěkná" URL, přičemž spoléhají na webový server, aby tato „pěkná" URL přeložil interně do takové podoby, kterou očekává daná webová aplikace.

Pravidla pro mod_rewrite je možné specifikovat jak v rámci konkrétního virtuálního webu, tak v souboru.htaccess, který umožňuje (pokud je to povoleno) upravit konfiguraci webového serveru pro daný adresář a podadresáře. Tento soubor naleznete jako součást mnoha webových aplikací.

Modul rewrite není ve výchozí konfiguraci Apache v Debianu aktivován, je tedy třeba jej nejprve povolit, takto:

```
a2enmod rewrite
/etc/init.d/apache2 restart
```

Pro fungování modulu rewrite jsou stěžejní čtyři direktivy: RewriteEngine, RewriteBase, RewriteCond aRewriteRule. Aby přepis URL fungoval, je třeba jej nejprve zapnout, buď na úrovni konfigurace daného virtuálního webu, nebo na úrovni .htaccess:

```
RewriteEngine on
```

Poté je možné použít ostatních direktiv. Při nasazení webových aplikací na server budete patrně nastavovat hodnotu direktivy RewriteBase, která upraví výchozí URL pro samotný přepis. Kupříkladu, pokud budete instalovat aplikaci do /appl na web (tzn. URL aplikace bude např. http://www.example.org/appl), specifikujete jako hodnotu v RewriteBase právě /appl. Toto umístění pak bude tvořit základnu pro všechny přepisy, které specifikujete pomocí RewriteCond a RewriteRule.

Samotný přepis URL je realizován pomocí určité podmínky (nebo podmínek), za kterých přepis nastane (ty jsou specifikovány pomocí direktivy RewriteCond), a jednoho nebo více pravidel pro samotný přepis (k tomu slouží direktiva RewriteRule). Syntax a možnosti obou direktiv jsou velice bohaté, já se omezím pouze na jediný příklad a odkážu vás v odkazech pod článkem na další zdroje, kde se dozvíte více.

```
RewriteCond  %{HTTP_HOST}  ^([^.]+)\.([^.]+)$
RewriteRule  ^(.*)$ http://www.%{HTTP_HOST}$1 [L,R=301,QSA]
```

Výše uvedené pravidlo demonstruje, že mod_rewrite můžete použít nejenom k zajištění pěkných URL, ale pro řadu jiných situací. Toto pravidlo vytvoří přesměrování na doménový tvar s „www", pokud klient zadá do prohlížeče pouze doménu druhého řádu. Tzn. pokud klient zadá do prohlížeče třeba "example.com/index.html", toto pravidlo prohlížeči vrátí přesměrování (HTTP kód 301) na URL "http://www.example.com/index.html". V podmínce vidíte na prvním místě definici proměnné k porovnání, v tomto případě HTTP_HOST, serverová proměnná obsahující hostname v HTTP požadavku. Hodnota se porovnává s regulárním výrazem (v syntaxi Perlu) specifikovaným jako druhý parametr RewriteCond. Tento regulární výraz zachytí hostname obsahující pouze jednu tečku, tedy doménu druhého řádu. Pokud klient specifikuje libovolnou doménu třetího řádu, pak pravidlu nevyhoví a přepis se neprovede.

Samotné pravidlo pro přepis specifikované v RewriteRule obsahuje na prvním místě regulární výraz, který bude aplikován na vstupní hodnotu RewriteRule, což je u prvního pravidla samotná URL cesta (cesta relativní k DocumentRoot), u dalších pravidel by to pak byl výstup z předchozího pravidla. V příkladu výše by to bylo /index.html (z http://example.com/index.html). Na druhém místě je řetězec pro substituci, který nahrazuje původní URL cestu. Na třetím místě, v hranatých závorkách, jsou značky (flags). Ty jsou zde tři, a sice značku „L", která označuje poslední pravidlo, tzn. po tomto přepisu se již nebudou provádět žádná další pravidla, dále značku „R=301" označující přesměrování pomocí kódu 301 (moved permanently), a QSA (query string append), která zajistí, že se na konec URL připojí i „query string", tj. případný kus zdrojového URL začínajícího otazníkem. Dolar a jednička v URL se vztahuje k regulárnímu výrazu, který je aplikován na vstupní hodnotu RewriteRule, zde bude odpovídat celé cestě relativní k DocumentRoot, tedy v příkladu výše /index.html.

### *mod_chroot*

Tento modul je jednou z možností, jak do jisté míry zvýšit bezpečnost. Funguje tak, že uzavře Apache v chrootu, tedy v adresáři, který si zvolíte a který se pro Apache stane kořenovým adresářem. V takovém případě, i kdyby se útočník dostal přes nějakou zranitelnost v Apachi dál, bude uzavřen v izolované oblasti, ze které se nedostane do zbytku systému (dostane se ovšem k webovým prezentacím). Tolik teorie, i když je rozhodně nutné dodat, že bezpečnost chrootu není v žádném případě neomezená (např. proces s rootovskými právy se může z chrootu dostat).

Příslušný modul naleznete v balíčku libapache2-mod-chroot, po jehož instalaci jej aktivujte obvyklým způsobem:

```
a2enmod mod_chroot
/etc/init.d/apache2 restart
```

V tuto chvíli se kromě samotného zavedení modulu ještě nic nestalo, neboť chroot je potřeba nejprve nakonfigurovat, a to direktivou ChrootDir, kterou můžete umístit třeba do /etc/apache2/httpd.conf. Jejím parametrem je adresář, který se pro Apache stane kořenovým:

```
ChrootDir /var/www
```

Při implementaci chrootu se budete potýkat s řadou problémů, které jsou s ním spojené. Prvním problémem je umístění souboru s PID Apache, který Apache umisťuje do /var/run. Pokud se však kořenovým adresářem stal /var/www, pak se Apache snaží umístit tento soubor do /var/www/var/run (který ve vašem systému patrně neexistuje). Tento adresář je tedy potřeba vytvořit.

Dalším problémem je nutnost změny konfigurace virtuálních webů, zejména veškerých direktivDocumentRoot. Možným rychlým řešením je vytvoření symbolického odkazu /var/www/var/www, který bude ukazovat na kořenový adresář, tedy /:

```
ln -s / /var/www/var/www
```

Po vyřešení těchto dvou problémů by se vám mělo podařit Apache spustit a vaše weby by měly fungovat, tedy alespoň ty statické, popřípadě ty, které využívají mod_php. Pokud budete chroot takto provozovat, jistě se setkáte se situací, kdy vaší webové aplikaci bude chybět něco, co se nachází mimo chroot. V té chvíli vám nezbude než potřebné soubory překopírovat do chrootu.

Posledním problémem, na který vás upozorním, je problém s ukončováním a restartem Apache. Je to dáno tím, že PID soubor se přesunul, ale init-skript v /etc/init.d/apache2 jej očekává ve /var/run. To můžete vyřešit opět symbolickým odkazem:

```
ln -s /var/www/var/run/apache2.pid /var/run/apache2.pid
```

V zásadě, použití chrootu má potenciál zvýšit bezpečnost vámi provozovaného webového serveru, ale za cenu možných problémů, na které patrně narazíte, pokud neprovozujete pouze statické weby. Na úplný závěr ještě zmíním alternativy, kterými je jistě mechanismus suexec a bezpečnostní řešení na úrovni jádra, jako je SELinux, AppArmor, Tomoyo apod.

## Správa linuxového serveru: Monitorování serveru pomocí nástroje Munin

Monitorování serveru zahrnuje mimo jiné i nutnost sledovat změny různých veličin v čase. Právě k tomu slouží nástroj Munin, který umožňuje zaznamenávat hodnoty mnoha systémových proměnných a vytvářet z nich grafy. Ty vám pak mohou pomoci nejenom odhalit řadu problémů, ale také odhadnout, kdy serveru přidat RAM, kdy nestačí dostupný výkon atd.

### Úvod

Munin je jedním z řady nástrojů, které umožňují sledovat různé systémové veličiny (obsazení RAM, zátěž procesoru, zatížení databáze, průběh e-mailů frontou atd.) v čase. Ze získaných dat vytvoří sadu HTML stránek s grafy, které mohou být zpřístupněny přes běžící webový server. Munin je modulární aplikací, nabízí řadu modulů, které zajišťují sběr specifických veličin (MySQL dotazy, Apache procesy atd.). Je samozřejmě možné si napsat vlastní moduly pro sběr dat, které Munin sbírat neumí. Munin je koncipován také jako síťový nástroj - umožňuje sbírat informace ze vzdálených zdrojů, a shromažďovat tak data z mnoha serverů na jednom místě k pohodlnému prohlížení. Každý generovaný graf je k dispozici v několika variantách - průběh hodnot během dne, týdne, měsíce a roku. Generované grafy mohou vypadat třeba takto (obrázek pochází z produkčního virtuálního serveru):

*Výstup Muninu na produkčním serveru, obsazení paměti a swapu*

### Instalace a zprovoznění

Instalace Muninu je velmi prostá. V případě Debianu postačí nainstalovat dva balíčky, a sice `munin` a `munin-node`:

```
aptitude install munin munin-node
```

První z uvedených balíčků obsahuje „serverovou část" nástroje, spouštěnou cronem, která sbírá údaje ze všech uzlů prostřednictvím démona (`munin-node`), který na daných uzlech běží. Výchozí konfigurace v Debianu je provozuschopná, tzn. po výše uvedeném kroku je již Munin zprovozněn, monitoruje danou stanici a generuje grafy do `/var/cache/munin/www` (toto umístění platí pro Debian Squeeze, v Debianu Lenny je to `/var/www/munin`).

I když to není nutné, doporučuji si hlavní konfigurační soubory projít a upravit podle svých představ. Stěžejní jsou dva, sice `/etc/munin/munin.conf`, který řídí „serverovou část", a `/etc/munin/munin-node.conf`, který řídí „klientského" démona běžícího na monitorovaném serveru. Určitě zkontrolujte `munin-node.conf` a konfigurační volbu `host`, která určuje, kde má démon naslouchat. Dodám, že tento démon běží s oprávněními uživatele root. Ve výchozím nastavení naslouchá na všech síťových rozhraních, což se hodí pouze u vzdálených uzlů, ze kterých chcete získávat údaje. U místního uzlu to není nutné, proto je vhodné konfiguraci upravit, aby démon naslouchal pouze na místní smyčce, takto:

```
# Which address to bind to;
#host *
host 127.0.0.1
```

### Přístup přes webový server

Grafy Muninu mohou obsahovat některé citlivější údaje, které mohou být využity případnými útočníky. I když veřejná dostupnost generovaných grafů na dobře zabezpečeném serveru nemusí představovat velké bezpečnostní riziko, je lepší přístup k nim zabezpečit, a to na úrovni webového serveru. V případě Apache můžete použít např. omezení na určité IP adresy naznačené níže:

```
Alias /munin /var/cache/munin/www
<Directory /var/cache/munin/www>
Order allow,deny
Allow from localhost 127.0.0.0/8 ::1
Allow from 1.2.3.4
Deny from all
Options None
</Directory>
```

Jinou možností může být HTTP autentizace a související nutnost se prokázat jménem a heslem. Z bezpečnostního hlediska je dobré takovou záležitost realizovat na virtuálním webu (virtual host) chráněném SSL a ujistit se, že na HTTP virtuálních webech je daný adresář nepřístupný. V Debianu Lenny se HTML výstup Muninu generuje do `/var/www/munin`, který je ve výchozí konfiguraci webových serverů veřejně přístupný, tudíž zde je vhodné přístup omezit. V Debianu Squeeze se výstupy generují do `/var/cache/munin/www` a do konfigurace Apache, pokud je nainstalovaný, je automaticky přidán konfigurační soubor `/etc/munin/apache.conf`, který tento adresář zpřístupňuje (pod aliasem `/munin`) takovým způsobem, že přístup k výstupu Muninu odjinud než z místní smyčky (localhost) je zakázán. Zde je tedy třeba povolit přístup z počítačů, odkud chcete k výstupům Muninu přistupovat.

### Práce s moduly

Moduly, které máte k dispozici, naleznete v `/usr/share/munin/plugins`. Moduly, které jsou aktivní, tj. které bude Munin (resp. `munin-node`) brát v úvahu při svém běhu, naleznete v `/etc/munin/plugins`. Aktivaci libovolného pluginu provedete vytvořením symbolického odkazu nebo prostým překopírováním. Některé moduly mohou vyžadovat dodatečné údaje, např. název rozhraní či databáze. Obvykle je to řešeno prakticky přejmenováním. Tzn. pokud byste chtěli monitorovat konkrétní síťové rozhraní, použijete modulif_. Abyste jej spojili s příslušným rozhraním, např. eth1, vytvoříte symbolický odkaz nebo kopii v`/etc/munin/plugins`, která se bude jmenovat if_eth1. Podobně je to řešeno třeba s monitorováním konkrétní databáze atd.

Některé pluginy mohou vyžadovat dodatečnou konfiguraci, která je pak realizována v`/etc/munin/conf.d/plugin-conf.d`, kde do libovolného souboru zapíšete příslušnou konfiguraci. Kupříkladu moduly pracující s databází PostgreSQL vyžadují konfigurační volbu v podobě uživatele, pod kterým se mají připojit k databázovému serveru (v Debianu je to uživatel „postgres"). Tato konfigurace by vypadala takto:

```
[postgres_*]
user postgres
```

V hranatých závorkách je název konkrétního modulu, resp. souboru (či souborů). Následují konfigurační volby. Ke zjištění, které volby jsou pro daný modul dostupné, popřípadě jak máte modul použít, se podívejte přímo na jeho „zdrojový kód", kde se kromě kódu nachází i dokumentace. Samotný modul je pak obyčejným spustitelným souborem - nejčastěji se jedná o shellové či perlové skripty. Samotné psaní vlastních modulů není nijak těžké, pokud umíte psát shellové skripty, zvládnete vytvořit vlastní modul. V odkazech pod článkem naleznete odkaz na příslušné HOWTO.

### Příjem dat z více serverů

Chcete-li shromažďovat data z více serverů na jednom počítači, postupujte následovně. Na všechny počítače, které chcete sledovat nainstalujte `munin-node`. Jelikož ve výchozím stavu tento démon naslouchá na síti, postačí už jen povolit přístup z „centrálního" počítače. To provedete přidáním volby allow do`/etc/munin/munin-node.conf`:

```
allow ^1\.2\.3\.4$
```

Jak je vidět, parametrem volby allow není IP adresa, ale regulární výraz. Výchozí konfigurace povoluje pouze přístup z místní smyčky (localhostu). Pokud chcete, démony můžete ještě navíc ochránit pomocíiptables. Tito démoni poslouchají na portu 4949. Posledním krokem je úprava nastavení Muninu, který běží na centrálním počítači, tedy souboru`/etc/munin/munin.conf`, kam je třeba přidat nastavení pro každý z monitorovaných počítačů. Toto nastavení může vypadat třeba takto:

```
[stroj1.example.org]
address 1.2.3.4
use_node_name yes
[stroj2.example.org]
address 4.3.2.1
use_node_name yes
```

Poté už jen vyčkejte a zkontrolujte, že se grafy generují pro všechny monitorované počítače.

## Správa linuxového serveru: Hlídání serverů pomocí nástroje Monit

Stalo se vám někdy, že vám na serveru spadl nějaký důležitý démon, popřípadě že server samotný přestal reagovat, a vy jste to nezjistili včas? Nástroj Monit umí hlídat činnost serveru i nejrůznějších démonů a nejen upozornit v případě problému, ale i provést opravnou akci. Jak na to, to je předmětem tohoto článku.

### Úvod

Monit je komplexní nástroj na sledování služeb, a to jak služeb běžících na místním serveru, tak služeb běžících na vzdáleném. Umožňuje velmi přesně definovat, co a jak se bude kontrolovat. U místních služeb může být prospěšný jak tím, že upozorní na jejich nedostupnost či blížící se problém (např. docházející paměť apod.), tak provedením opravné akce, kterou může být restart monitorované služby, ale také jakýkoliv jiný příkaz. Monitorovat lze nejen činnost konkrétních démonů, ale také řadu systémových veličin, jako je např. zatížení systému, volná paměť, využití swapu apod.

Stav monitorovaných systémů či služeb je možné zjišťovat nejenom kontrolou poštovní schránky, kam přichází oznámení, ale také pomocí webového či shellového rozhraní. Tyto rozhraní umí nejenom sledovat stav veškerého monitorování, ale umožňují i provádět jisté změny přímo za běhu, tedy restart sledovaných služeb nebo pozastavení či opětovné zahájení konkrétního monitorování. Na to je dobré při případných upgradech či servisních zásazích nezapomenout, protože Monit pak třeba službu, která není aktivní, automaticky nastartuje, což v takových situacích může vadit.

### Instalace

K instalaci postačí nainstalovat stejnojmenný balíček, v Debianu stačí zapsat následující příkaz:

```
aptitude install monit
```

Na rozdíl od řady jiných démonů je Monit po instalaci neaktivní, tzn. před samotným použitím je třeba jej nakonfigurovat (viz níže). Až budete mít Monit nakonfigurovaný, povolte jeho použití v /etc/default/monit, kde proměnné startup přiřaďte jedničku místo nuly:

```
startup=1
```

### Základní konfigurace Monitu

Proces konfigurace Monitu je poměrně jednoduchý. Máte na výběr ze dvou možností - buď budete přímo měnit hlavní konfigurační soubor Monitu, tedy /etc/monit/monitrc, nebo si jednotlivé služby i základní konfiguraci rozdělíte do libovolně pojmenovaných souborů v /etc/monit/conf.d.

Výchozí konfigurační soubor /etc/monit/monitrc neobsahuje prakticky žádnou výchozí konfiguraci, ale je dobře zdokumentovaný a základní nastavení jsou v něm přítomna, pouze jsou zakomentována, takže stačí příslušné volby odkomentovat a nastavit si je podle svých představ. Základ by mohl vypadat třeba takto:

```
set daemon 120
with start delay 240
set logfile syslog facility log_daemon
set mailserver localhost
set mail-format { from: monit@muj-server.tld }
set alert root@localhost
```

První dva řádky nastavují režim démona a dva podstatné časové údaje. Prvním z nich je doba cyklu, tedy doba mezi jednotlivými kontrolami, zde nastavena na sto dvacet sekund. V jednotlivých pravidlech (viz dále) je možné stav služeb zjišťovat i v delším časovém rozmezí a konkrétní akci pak vázat na stav, který trvá po dobu několika cyklů.

Na druhém řádku je nastavena úvodní čekací doba před zahájením monitorování. Tato doba se měří od spuštění Monitu a v tomto případě je nastavena na dvě stě čtyřicet sekund. Druhý řádek je samozřejmě nepovinný - je to pouze doplnění k prvnímu řádku, a pokud jej nespecifikujete, služby se začnou monitorovat okamžitě po spouštění Monitu.

Třetí řádek nastavuje logování, čtvrtý pak poštovní servery (mailserver), prostřednictvím kterých budou odesílána oznámení. Mailserverů je možné specifikovat více, jako oddělovač použijte čárku. Máte-li možnost, nelze než doporučit použít víc než jeden poštovní server, Monit pak v případě nečinnosti jednoho pošle e-mail přes jiný (je nepříjemné, pokud by přestal fungovat poštovní server a vy byste se o tom nedozvěděli, protože Monit by neměl k dispozici žádný další). Pátý řádek upravuje formát zasílaných e-mailů, resp. nastavuje pole From: - to bývá vhodné přizpůsobit doménovému jménu vašeho serveru.

Poslední řádek nastavuje příjemce upozornění. Příjemců může být pochopitelně více - potřebujete-li jich více, specifikujte set alert vícekrát, např. takto:

```
set alert admin1@localhost
set alert admin2@localhost
```

Dodám ještě, že Monit umožňuje zasílat různé typy zpráv různým uživatelům, avšak zde vás už odkážu nadokumentaci nebo alespoň na drobný příklad v komentovaném monitrc.

### Monitorování systému

Pokud chcete monitorovat základní veličiny systému, na kterém Monit běží, můžete použít následující šablonu:

```
check system muj-server.tld
if loadavg (1min) > 4 then alert
if loadavg (5min) > 2 then alert
if memory usage > 75% then alert
if cpu usage (user) > 70% then alert
if cpu usage (system) > 30% then alert
if cpu usage (wait) > 20% then alert
```

Název systému a hodnoty si samozřejmě upravte podle libosti. Jak je vidět, zápis pravidel pro Monit se podobá konstrukcím programovacích jazyků, je ovšem maximálně zjednodušený a dobře čitelný. Akce „alert" zde představuje odeslání zprávy, pokud bude daná podmínka splněna.

### Monitorování služeb

Asi nejjednodušší je poohlédnout se po šablonách pro konkrétní služby, které chcete monitorovat. Něco málo je k dispozici v návodech, ale obrovská spousta příkladů se nachází v příkladech na webu projektu. Využijete-li některé z těchto šablon, nezapomeňte si je upravit, tedy přinejmenším se ujistit, že umístění všech souborů odpovídá vašemu systému. Zde se jedná zejména o init skripty, které jsou v Debianu a mnoha distribucích umístěny v /etc/init.d, ale třeba Arch Linux je má v /etc/rc.d. Dále se jedná o umístění případných socketů či souborů s PID příslušných démonů, kde se umístění mezi distribucemi může také lišit.

Příklad monitorované služby by mohl vypadat třeba takto:

```
check process denyhosts with pidfile /var/run/denyhosts.pid
start program = "/etc/init.d/denyhosts start"
stop program = "/etc/init.d/denyhosts stop"
if cpu > 60% for 2 cycles then alert
if cpu > 80% for 5 cycles then restart
if totalmem > 15.0 MB for 5 cycles then restart
```

V tomto příkladu je monitorován proces Denyhosts (nástroj byl v tomto seriálu již představen) za pomoci souboru s PID (with pidfile ...). Je-li specifikován PID soubor, Monit hlídá, jestli existuje proces s příslušným číslem, které je v něm uloženo. Na druhém a třetím řádku jsou specifikovány příkazy pro pro spuštění a zastavení démona, které jsou využívány pro řízení běhu služeb. Monit může příslušné služby restartovat, nastartovat nebo i zastavit za vámi specifikovaných okolností. Na čtvrtém a pátém řádku jsou podmínky vztahující se k zatížení procesoru samotným Denyhosts. Ano, Monit může sledovat využití systémových prostředků (CPU, paměti atd.) u konkrétních procesů, a na to mohou být navázány podmínky.

Pokud by Denyhosts během dvou cyklů (tedy dvou kontrol) zjistil, že Denyhosts využívá více než šedesát procent CPU, zaslal by e-mailem upozornění. Pokud by Denyhosts využíval více než osmdesát procent CPU během pěti cyklů, byl by poté proveden restart služby. Poslední řádek je pojistka proti případným memory leakům, tzn. v případě obsazení více než 15 MB operační paměti po dobu pěti cyklů bude proveden restart.

Tím bych tento díl ukončil. Příště budou následovat další příklady, budou naznačeny možnosti seskupování procesů a konfigurace a práce s webovým a konzolovým rozhraním Monitu.

## Správa linuxového serveru: Hlídání pomocí Monitu (pokračování)

Minulý díl představil nástroj Monit a jeho základní konfiguraci, tento díl půjde o kus dále a ukáže vám některé další příklady konfigurace Monitu, včetně monitorování služeb běžících na jiných počítačích. Chybět nebude ani představení webového a konzolového rozhraní Monitu.

### Další příklady konfigurace hlídání služeb

Hlídání např. webového serveru se nemusí omezit na pouhou kontrolu, zdali server odpovídá na portu 80, popřípadě 443. Je možné si vyžádat a prověřit i konkrétní URL na konkrétní doméně, jak je naznačeno níže:

```
check process apache with pidfile /var/run/apache2.pid
start "/etc/init.d/apache2 start"
stop  "/etc/init.d/apache2 stop"
if failed host www.domena.tld port 80 protocol http
and request "/doku.php"
with timeout 15 seconds
then restart
if cpu > 60% for 2 cycles then alert
if cpu > 80% for 5 cycles then restart
if totalmem > 600.0 MB for 5 cycles then restart
if children > 250 then restart
if 3 restarts within 5 cycles then timeout
group lamp
```

Jednotlivé služby je možné seskupovat do celků, se kterými se pak dá zacházet najednou, tzn. provést restart, zastavit služby, vypnout či zapnout jejich hlídání atd. Této vlastnosti využijete především při práci s konzolovým obslužným nástrojem. Službu zařadíte do skupiny pomocí direktivy group. V příkladu výše je hlídaný Apache členem skupiny „lamp".

Monit umožňuje sledovat i jednotlivé konfigurační soubory, umí je přiřadit ke konkrétní službě a umí dokonce reagovat na jejich změny:

```
check process apache with pidfile /var/run/apache2.pid
start "/etc/init.d/apache2 start"
stop  "/etc/init.d/apache2 stop"
[...]
group server
depends on httpd.conf

check file httpd.conf
with path /etc/apache/httpd.conf
if changed checksum
then exec "/usr/sbin/apache2ctl graceful"
```

Zde je na změnu httpd.conf navázán automatický restart Apache. Je to samozřejmě pouze příklad, v praxi bych se asi k něčemu takovému neuchyloval, abych pak náhodou při nějaké rozsáhlejší změně konfiguračních souborů uprostřed práce nezjistil, že mi Monit na pozadí restartoval Apache, který se samozřejmě kvůli chybě v syntaxi konfigurace znovu nespustil.

V tomto příkladu je však patrná ještě jedna direktiva, a sice direktiva závislosti, tedy „depends on". Tímto způsobem je možné vytvářet mezi jednotlivými hlídanými službami závislostní vazby, které pak Monit při všech operacích bere v úvahu.

### Monitorování vzdálených systémů a služeb

Pomocí Monitu lze však hlídat nejenom lokálně běžící služby, ale také vzdálené služby a počítače v síti. Můžete tak z jednoho místa monitorovat stav všech svých serverů. Jednoduchý příklad hlídání vzdáleného serveru následuje:

```
check host domena.tld with address domena.tld
if failed icmp type echo count 3 with timeout 15 seconds for 5 cycles then alert
if failed port 25 proto smtp for 5 cycles then alert
if failed port 80 proto http for 5 cycles then alert
if failed port 22 proto ssh for 5 cycles then alert
```

Základem pro monitorování vzdálených serverů je jejich dostupnost na síti, což lze testovat prostřednictvím protokolu ICMP (viz druhý řádek příkladu). Zde, pokud na ping vzdálený server neodpoví během pěti cyklů Monitu, zašle se o tom upozornění správci. Hlídání je pak nastaveno tak, že se v každém cyklu zasílají celkem tři ICMP pakety a na odpověď se čeká po dobu patnácti sekund.

Hlídání dalších služeb pak následuje (řádky 3-5). V tomto případě se jedná o testování fungování příslušného protokolu, nikoliv tedy o pouhý test TCP spojení. Monit podporuje celou řadu protokolů, viz tatočást oficiální dokumentace.

### Webové rozhraní Monitu

Stav služeb a jejich správu je možné realizovat nejenom prostřednictvím konzolového nástroje (viz dále), ale také pohodlně pomocí webového rozhraní. V případě Monitu není třeba používat webový server k jeho zobrazení, ten si Monit zajišťuje sám, stačí tedy toto rozhraní pouze nakonfigurovat. Nastavení webového rozhraní může vypadat například takto:

```
set httpd port 2812 and
ssl enable
pemfile /etc/cert/server.pem
use address localhost
allow localhost
allow admin:monit
```

Toto nastavení využívá SSL s certifikátem umístěným v /etc/cert/server.pem. Webový server bude v tomto případě Monitu naslouchat pouze na localhostu (viz čtvrtý řádek), bude také povolovat spojení pouze z localhostu (pátý řádek) a bude vyžadovat HTTP autentzaci s uživatelským jménem „admin" a heslem „monit" (doporučuji změnit na něco složitějšího a ujistit se, že soubor s konfigurací Monitu nepřečte obyčejný uživatel na serveru).

Abyste získali představu, jak toto rozhraní vypadá, podívejte se na následující snímky obrazovek. Předem se omlouvám za nutnou cenzuru některých položek. Základní obrazovka webového rozhraní Monitu vypadá následovně:


*Základní obrazovka webového rozhraní Monitu*

Pokud byste se podívali na detail některé hlídané služby, vypadalo by to takto:


*Detail hlídané služby*

Jak je z obrázku patrné, službu je možné ovládat pomocí tlačítek ve spodní části obrazovky. Snadno tak zajistíte vzdálený restart bez nutnosti použít příkazovou řádku. Detail hlídání vzdáleného serveru může vypadat kupříkladu takto:


*Detail hlídaného vzdáleného serveru*

Otázkou zůstává, jak co nejbezpečněji zpřístupnit rozhraní Monitu zvenčí. Jednou z možností je použít SSH tunel, např. takto:

```
ssh -N -f uzivatel@server -L 3000:127.0.0.1:2812
```

Tento příkaz naváže SSH spojení se serverem „server" jako uživatel „uzivatel" a nabídne vám na místním portu 3000 vašeho počítače vzdálený port Monitu běžící na portu 2812 na IP adrese 127.0.0.1 (tedy na localhostu serveru), tzn. do prohlížeče pak stačí zapsat: https://localhost:3000/ a máte k dispozici webové rozhraní Monitu, se kterým komunikujete bezpečně přes SSH tunel.

Pokud vám na serveru běží webový server s PHP, můžete se podívat do /usr/share/doc/monit/contrib(platí pro Debian a z něj odvozené distribuce), kde se nachází soubor monit.php, který umožňuje zpřístupnit webové rozhraní Monitu prostřednictvím PHP kdekoliv na serveru. Pokud se rozhodnete pro tuto cestu, přístup k tomuto souboru určitě dobře zabezpečte.

## Konzolové rozhraní Monitu

Funguje-li Monit jako démon, můžete využít řádkového nástroje monit ke zjišťování stavu služeb a servisním zásahům. Můžete si, kupříkladu, vypsat stav všech hlídaných služeb:

```
debian:~# monit summary
The Monit daemon 5.1.1 uptime: 19d 4h 31m

System 'domena.tld'                 running
Process 'apache'                    running
Process 'postgresql'                running
Process 'sshd'                      running
Process 'postfix'                   running
Process 'denyhosts'                 running
Process 'dkim-filter'               not monitored
Remote Host 'domena1.tld'           online with all services
Remote Host 'domena2.tld'           ICMP failed
Remote Host 'domena3.tld'           online with all services
```

Z výpisu je patrná jedna nemonitorovaná služba a nedostupnost jednoho vzdáleného serveru (domena2.tld). Základních operací s jednotlivými službami je pět, a sice jejich nastartování (start), zastavení (stop), restart (restart), zahájení hlídání (monitor) a přerušení hlídání (unmonitor). Jako parametr kterékoliv z těchto akcí můžete specifikovat jméno konkrétní služby, nebo zvolit všechny služby pomocí parametru „all". Zastavení Apache by vypadalo takto:

```
monit stop apache
```

Zajímavější je možnost využívání skupin (viz výše), tedy možnost provést konkrétní akci pro všechny služby z dané skupiny. Pokud byste měli skupinu „lamp" a chtěli všechny její služby restartovat, provedli byste to takto:

```
monit -g lamp restart all
```

Ostatní volby a možnosti tohoto nástroje si můžete prohlédnout zadáním:

```
monit -h
```

## Velmi flexibilní nástroj na hlídání systému

Monit je velmi flexibilní nástroj na hlídání systému, služeb i vzdálených počítačů. Nabízí velmi široké možnosti konfigurace, z nichž tu byl ukázán spíše jen nutný základ. Pokud se chcete dozvědět více, určitě nahlédněte do odkazů pod článkem, kde naleznete nejen oficiální dokumentaci, ale také wiki s řadou příkladů konfigurace služeb.

## Správa linuxového serveru: Hlídání pomocí Monitu (pokračování)

Minulý díl představil nástroj Monit a jeho základní konfiguraci, tento díl půjde o kus dále a ukáže vám některé další příklady konfigurace Monitu, včetně monitorování služeb běžících na jiných počítačích. Chybět nebude ani představení webového a konzolového rozhraní Monitu.

### Další příklady konfigurace hlídání služeb

Hlídání např. webového serveru se nemusí omezit na pouhou kontrolu, zdali server odpovídá na portu 80, popřípadě 443. Je možné si vyžádat a prověřit i konkrétní URL na konkrétní doméně, jak je naznačeno níže:

```
check process apache with pidfile /var/run/apache2.pid
start "/etc/init.d/apache2 start"
stop  "/etc/init.d/apache2 stop"
if failed host www.domena.tld port 80 protocol http
and request "/doku.php"
with timeout 15 seconds
then restart
if cpu > 60% for 2 cycles then alert
if cpu > 80% for 5 cycles then restart
if totalmem > 600.0 MB for 5 cycles then restart
if children > 250 then restart
if 3 restarts within 5 cycles then timeout
group lamp
```

Jednotlivé služby je možné seskupovat do celků, se kterými se pak dá zacházet najednou, tzn. provést restart, zastavit služby, vypnout či zapnout jejich hlídání atd. Této vlastnosti využijete především při práci s konzolovým obslužným nástrojem. Službu zařadíte do skupiny pomocí direktivy group. V příkladu výše je hlídaný Apache členem skupiny „lamp".

Monit umožňuje sledovat i jednotlivé konfigurační soubory, umí je přiřadit ke konkrétní službě a umí dokonce reagovat na jejich změny:

```
check process apache with pidfile /var/run/apache2.pid
start "/etc/init.d/apache2 start"
stop  "/etc/init.d/apache2 stop"
[...]
group server
depends on httpd.conf

check file httpd.conf
with path /etc/apache/httpd.conf
if changed checksum
then exec "/usr/sbin/apache2ctl graceful"
```

Zde je na změnu httpd.conf navázán automatický restart Apache. Je to samozřejmě pouze příklad, v praxi bych se asi k něčemu takovému neuchyloval, abych pak náhodou při nějaké rozsáhlejší změně konfiguračních souborů uprostřed práce nezjistil, že mi Monit na pozadí restartoval Apache, který se samozřejmě kvůli chybě v syntaxi konfigurace znovu nespustil.

V tomto příkladu je však patrná ještě jedna direktiva, a sice direktiva závislosti, tedy „depends on". Tímto způsobem je možné vytvářet mezi jednotlivými hlídanými službami závislostní vazby, které pak Monit při všech operacích bere v úvahu.

### Monitorování vzdálených systémů a služeb

Pomocí Monitu lze však hlídat nejenom lokálně běžící služby, ale také vzdálené služby a počítače v síti. Můžete tak z jednoho místa monitorovat stav všech svých serverů. Jednoduchý příklad hlídání vzdáleného serveru následuje:

```
check host domena.tld with address domena.tld
if failed icmp type echo count 3 with timeout 15 seconds for 5 cycles then alert
if failed port 25 proto smtp for 5 cycles then alert
if failed port 80 proto http for 5 cycles then alert
if failed port 22 proto ssh for 5 cycles then alert
```

Základem pro monitorování vzdálených serverů je jejich dostupnost na síti, což lze testovat prostřednictvím protokolu ICMP (viz druhý řádek příkladu). Zde, pokud na ping vzdálený server neodpoví během pěti cyklů Monitu, zašle se o tom upozornění správci. Hlídání je pak nastaveno tak, že se v každém cyklu zasílají celkem tři ICMP pakety a na odpověď se čeká po dobu patnácti sekund.

Hlídání dalších služeb pak následuje (řádky 3-5). V tomto případě se jedná o testování fungování příslušného protokolu, nikoliv tedy o pouhý test TCP spojení. Monit podporuje celou řadu protokolů, viz tatočást oficiální dokumentace.

### Webové rozhraní Monitu

Stav služeb a jejich správu je možné realizovat nejenom prostřednictvím konzolového nástroje (viz dále), ale také pohodlně pomocí webového rozhraní. V případě Monitu není třeba používat webový server k jeho zobrazení, ten si Monit zajišťuje sám, stačí tedy toto rozhraní pouze nakonfigurovat. Nastavení webového rozhraní může vypadat například takto:

```
set httpd port 2812 and
ssl enable
pemfile /etc/cert/server.pem
use address localhost
allow localhost
allow admin:monit
```

Toto nastavení využívá SSL s certifikátem umístěným v /etc/cert/server.pem. Webový server bude v tomto případě Monitu naslouchat pouze na localhostu (viz čtvrtý řádek), bude také povolovat spojení pouze z localhostu (pátý řádek) a bude vyžadovat HTTP autentzaci s uživatelským jménem „admin" a heslem „monit" (doporučuji změnit na něco složitějšího a ujistit se, že soubor s konfigurací Monitu nepřečte obyčejný uživatel na serveru).

Abyste získali představu, jak toto rozhraní vypadá, podívejte se na následující snímky obrazovek. Předem se omlouvám za nutnou cenzuru některých položek. Základní obrazovka webového rozhraní Monitu vypadá následovně:


*Základní obrazovka webového rozhraní Monitu*

Pokud byste se podívali na detail některé hlídané služby, vypadalo by to takto:


*Detail hlídané služby*

Jak je z obrázku patrné, službu je možné ovládat pomocí tlačítek ve spodní části obrazovky. Snadno tak zajistíte vzdálený restart bez nutnosti použít příkazovou řádku. Detail hlídání vzdáleného serveru může vypadat kupříkladu takto:


*Detail hlídaného vzdáleného serveru*

Otázkou zůstává, jak co nejbezpečněji zpřístupnit rozhraní Monitu zvenčí. Jednou z možností je použít SSH tunel, např. takto:

```
ssh -N -f uzivatel@server -L 3000:127.0.0.1:2812
```

Tento příkaz naváže SSH spojení se serverem „server" jako uživatel „uzivatel" a nabídne vám na místním portu 3000 vašeho počítače vzdálený port Monitu běžící na portu 2812 na IP adrese 127.0.0.1 (tedy na localhostu serveru), tzn. do prohlížeče pak stačí zapsat: https://localhost:3000/ a máte k dispozici webové rozhraní Monitu, se kterým komunikujete bezpečně přes SSH tunel.

Pokud vám na serveru běží webový server s PHP, můžete se podívat do /usr/share/doc/monit/contrib(platí pro Debian a z něj odvozené distribuce), kde se nachází soubor monit.php, který umožňuje zpřístupnit webové rozhraní Monitu prostřednictvím PHP kdekoliv na serveru. Pokud se rozhodnete pro tuto cestu, přístup k tomuto souboru určitě dobře zabezpečte.

## Konzolové rozhraní Monitu

Funguje-li Monit jako démon, můžete využít řádkového nástroje monit ke zjišťování stavu služeb a servisním zásahům. Můžete si, kupříkladu, vypsat stav všech hlídaných služeb:

```
debian:~# monit summary
The Monit daemon 5.1.1 uptime: 19d 4h 31m

System 'domena.tld'                 running
Process 'apache'                    running
Process 'postgresql'                running
Process 'sshd'                      running
Process 'postfix'                   running
Process 'denyhosts'                 running
Process 'dkim-filter'               not monitored
Remote Host 'domena1.tld'           online with all services
Remote Host 'domena2.tld'           ICMP failed
Remote Host 'domena3.tld'           online with all services
```

Z výpisu je patrná jedna nemonitorovaná služba a nedostupnost jednoho vzdáleného serveru (domena2.tld). Základních operací s jednotlivými službami je pět, a sice jejich nastartování (start), zastavení (stop), restart (restart), zahájení hlídání (monitor) a přerušení hlídání (unmonitor). Jako parametr kterékoliv z těchto akcí můžete specifikovat jméno konkrétní služby, nebo zvolit všechny služby pomocí parametru „all". Zastavení Apache by vypadalo takto:

```
monit stop apache
```

Zajímavější je možnost využívání skupin (viz výše), tedy možnost provést konkrétní akci pro všechny služby z dané skupiny. Pokud byste měli skupinu „lamp" a chtěli všechny její služby restartovat, provedli byste to takto:

```
monit -g lamp restart all
```

Ostatní volby a možnosti tohoto nástroje si můžete prohlédnout zadáním:

```
monit -h
```

## Velmi flexibilní nástroj na hlídání systému

Monit je velmi flexibilní nástroj na hlídání systému, služeb i vzdálených počítačů. Nabízí velmi široké možnosti konfigurace, z nichž tu byl ukázán spíše jen nutný základ. Pokud se chcete dozvědět více, určitě nahlédněte do odkazů pod článkem, kde naleznete nejen oficiální dokumentaci, ale také wiki s řadou příkladů konfigurace služeb.

## Správa linuxového serveru: PostgreSQL: Klienti a zálohování

Minulý díl se věnoval instalaci a zprovoznění PostgreSQL. Tento díl představí klienty pro správu a obsluhu Postgresu a zmíní zálohování a obnovu Postgresu.

### Řádkový klient psql a změna hesla uživatele

Po instalaci Postgresu na server budete mít k dispozici řádkového klienta psql. Pokud při jeho spuštění nepoužijete žádné parametry, pokusí se připojit k místní databázi jako uživatel, pod kterým jste přihlášeni. Uživatelem s administrátorskými právy je v případě Postgresu uživatel postgres, nikoliv „obvyklý" uživatelroot. Pokud jste uživateli postgres nenastavili vlastní heslo, budete muset získat přístup k DBMS bez hesla, a sice jako unixový uživatel postgres:

```
su postgres -
```

Poté můžete spustit klienta psql:

```
postgres@debian:/root$ psql
psql (8.4.7)
Pro získání nápovědy napište "help".

postgres=#
```

Chcete-li nastavit uživateli postgres nějaké heslo, abyste mohli k DBMS přistupovat jako databázový uživatel postgres bez nutnosti měnit unixového uživatele pomocí su, použijte buď interaktivní příkaz\password, nebo SQL příkaz ALTER ROLE:

```
postgres=# ALTER ROLE postgres WITH PASSWORD 'moje_dlouhe_a_bezpecne_heslo';
```

Budete-li chtít provádět administrátorské úkony třeba ze skriptů spuštěných s právy roota, můžete samozřejmě použít su mechanismu pro změnu efektivního uživatele na postgres, jen je třeba si dát pozor na to, že uživatel postgres má na rozdíl od roota právo zápisu pouze do některých adresářů (zejména/var/lib/postgres, kde sídlí konfigurační i datové soubory DBMS). Jinou možností je nastavit databázovému uživateli postgres heslo a přistupovat k němu pohodlně pod jakýmkoliv uživatelem. Zde ovšem přichází na řadu drobný problém, a sice jakým způsobem nástroji psql předat heslo, aby nebylo nutné jej pokaždé zadávat interaktivně. Za tímto účelem můžete použít soubor .pgpass, který vytvořte ve svém domovském adresáři se obsahem podle následujícího vzoru:

```
jmeno_pocitace:port:databaze:uzivatel:heslo
```

Konkrétní příklad pro Postgres běžící na localhostu na standardním portu, uživatele postgres a libovolnou databázi by mohl vypadat třeba takto:

```
localhost:5432:*:postgres:moje_dlouhe_a_bezpecne_heslo
```

Samotné přihlášení k DBMS pomocí psql pak provedete takto:

```
psql -U postgres -h localhost -p 5432
```

Parametr -U značí uživatele, -h název počítače, na kterém běží DB, a -p udává port (Postgres standardně běží na portu 5432).

### *Nouzový rychlokurz psql*

Jste-li zvyklí převážně na MySQL a na příkazy typu SHOW DATABASES či SHOW TABLES, které vám v psql určitě fungovat nebudou, hodí se vám tento nouzový rychlokurz klienta psql. Dodávám, že samotný nástroj psqlmá velmi dobrou dokumentaci, kterou určitě doporučuji prostudovat.

**Globální operace**

- \l - vypíše seznam databází a základních informací o nich
- \db - vypíše seznam tablespaces
- \dg - vypíše seznam rolí
- \dn - vypíše seznam schémat
- \password [uzivatel] - interaktivní nastavení hesla uživatele uzivatel; pokud nebude zadán uživatel, změna se bude týkat aktuálně přihlášeného uživatele
- \c [databaze] [uzivatel] [pocitac] [port] - připojí se k databázi

**Operace nad konkrétní databází**

- \di - vypíše seznam indexů
- \ds - vypíše seznam sekvencí
- \dt - vypíše seznam tabulek
- \dv - vypíše seznam pohledů (views)

### Webový klient phppgadmin

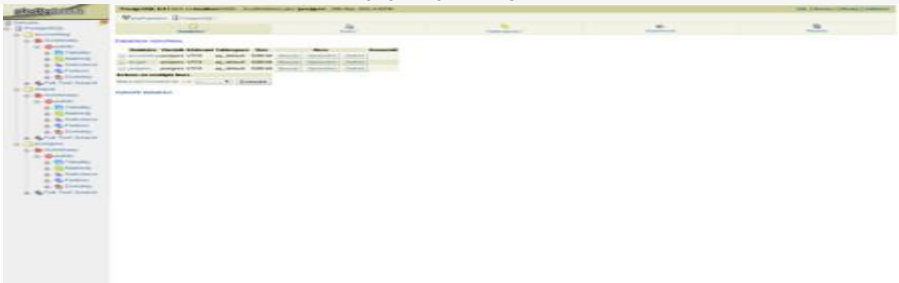
*phppgadmin: Uživatel postgres přihlášený ihned po instalaci*

Znáte-li MySQL, znáte také určitě vynikajícího webového klienta/správce phpMyAdmin. Ten je ovšem stavěn pouze pro MySQL a pro Postgres jej není možné použít. Existuje však jiný projekt, a sicephpPgAdmin, který se snaží kráčet ve stejných šlépějích jako phpMyAdmin, přičemž je stavěn pro Postgres. Jeho instalace v Debianu je velmi jednoduchá, stačí nainstalovat příslušný balíček:

```
aptitude install phppgadmin
```

Používáte-li webový server Apache, bude jeho konfigurace automaticky upravena při instalaci balíčku, a to přidáním symbolického odkazu phppgadmin do /etc/apache2/conf.d. Pro úplnost dodávám, že tento symbolický odkaz ukazuje na /etc/phppgadmin/apache.conf. Výchozí úprava konfigurace pro Apache je poměrně restriktivní, umožňuje pouze připojení z localhostu. Pokud budete chtít tento nástroj využívat k provádění správcovských činností, nemusí být od věci to tak nechat (a použít SSH tunel), popřípadě povolit přístup pouze z některých počítačů:

```
allow from 1.2.3.4
```



*phppgadmin: Pohled na nově vytvořené databáze*

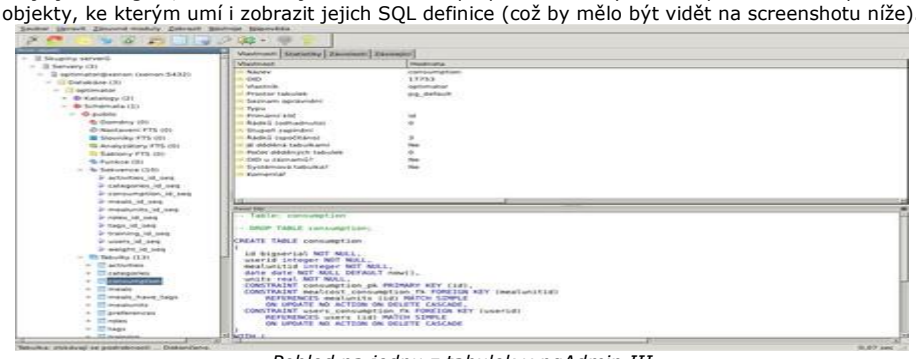
Samotné přihlášení administrátora je v phppgadmin rovněž zakázáno. To má na svědomí volbaextra_login_security v /etc/phppgadmin/config.inc.php:

```
$conf['extra_login_security'] = true;
```

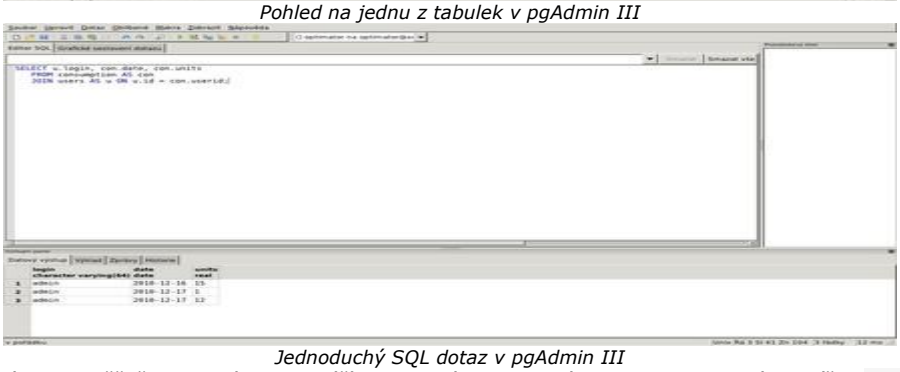
Jste-li si jisti, že chcete přihlášení správců opravdu povolit, je třeba tuto volbu nastavit na false.

## Desktopový GUI klient pgAdmin III

pgAdmin III je patrně nejlepší dostupný FOSS nástroj určený pro správu Postgres DBMS. Jedná se o multiplatformní desktopovou aplikaci, která umožňuje připojení k místním i vzdáleným databázím, jejich pohodlnou správu i klasický SQL přístup. SQL dotazy je možné zadávat ručně nebo je sestavit graficky. Podporuje jak Postgres, tak některé jeho komerční a proprietární klony. Je schopen zobrazit a pracovat s veškerými Postgres objekty, ke kterým umí i zobrazit jejich SQL definice (což by mělo být vidět na screenshotu níže).

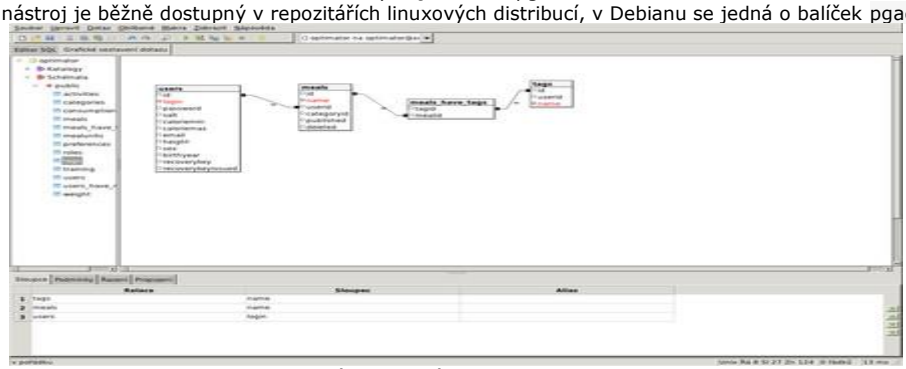


*Pohled na jednu z tabulek v pgAdmin III*



*Jednoduchý SQL dotaz v pgAdmin III*

Tento nástroj je běžně dostupný v repozitářích linuxových distribucí, v Debianu se jedná o balíček pgadmin3.



*Grafické sestavení dotazu v pgAdmin III*

Samotné připojení ke vzdálené databázi je možné provést řadou způsobů. Pro správce je patrně nejrychlejší použít SSH tunel, jehož vytvoření je popsáno v Úvodu do SSH pro správce. Je samozřejmě také možné nastavit Postgres, aby přijímal spojení ze sítě a umožňoval přihlášení uživatele či uživatelů z určité IP adresy. K tomu slouží konfigurační soubor pg_hba.conf, který v Debianu Squeeze s Postgresem 8.4 naleznete v /etc/postgresql/8.4/main/.

## Zálohování Postgresu

K zálohování konkrétní databáze se používá nástroj pg_dump, který má stejné parametry jako interaktivní klient psql. Zálohování databáze postfix dostupné na místním serveru a na standardním portu by se tedy s použitím uživatele postgres provedlo takto:

```
pg_dump -U postgres -h localhost -p 5432 postfix > zaloha_postfix.sql
```

Výstupem je obsah tabulky v SQL, který by se vypsal na standardní výstup (tedy obvykle přímo do konzole), proto bylo v tomto případě použito přesměrování do souboru zaloha_postfix.sql. V případě větší databáze by bylo vhodnější zapojit kompresi, například takto:

```
pg_dump -U postgres -h localhost -p 5432 postfix | gzip > zaloha_postfix.sql.gz
```

Chcete-li zazálohovat celý databázový cluster, použijete nástroj pg_dumpall, prakticky totožným způsobem jako pg_dump, pouze neuvedete název konkrétní databáze pro zálohování (zde už raději uvádím pouze příklad s kompresí):

```
pg_dump -U postgres -h localhost -p 5432 | gzip > zaloha_clusteru.sql.gz
```

## Obnova ze záloh

Tento článek začal představením klienta psql a tímto nástrojem také skončí. Jelikož záloha DB/DBMS má podobu sady SQL příkazů uložených v určitém souboru, není nic jednoduššího než vzít klienta psql a všechny SQL příkazy postupně provést, takto:

```
psql -U postgres -h localhost -p 5432 < zaloha.sql
```

V případě komprimovaného archívu si můžete pomoci rourou, takto:

```
gzip -c -d zaloha_clusteru.sql.gz | psql -U postgres -h localhost -p 5432
```

Nezapomeňte se přečíst náš seriál o optimalizaci PostgreSQL.

## Správa linuxového serveru: Sledování logů pomocí nástroje logcheck

Každý zodpovědný správce linuxového serveru sleduje, co se na serveru děje, zdali nedochází k nějakým problémům či zdali nedošlo k nějakému bezpečnostnímu incidentu. Sledování serveru zahrnuje kromě monitorování základních veličin i prohlížení a sledování logů. Jelikož ruční procházení megabytů logů není ani pohodlné, ani reálné, existují nástroje, které tuto činnost usnadňují. Jedním z těchto nástrojů, konkrétně pak nástrojem logcheck, se bude zabývat tento díl.

### Stručný úvod do logů

Jak jistě víte, logy jsou soubory obsahující hlášení různých komponent systému, počínaje jádrem až po hlášení jednotlivých démonů. Zaznamenávání systémových hlášení má na starosti démon syslog. Aby to nebylo jednoduché, existuje hned několik implementací syslogu. V Debianu je výchozí rsyslog, jehož konfigurační soubor sídlí v /etc/rsyslog.conf (a jeho doplňky v /etc/rsyslog.d). Zde se obvykle nastavuje především to, jaké hlášky putují do jakých logů. Samotné soubory s logy sídlí ve /var/log, přičemž jednotlivé záznamy bývají rozděleny do řady souborů kvůli snazšímu vyhledávání. Mezi podstatné systémové logy v Debianu patří messages, syslog, auth.log,debug, kern.log a mail.log. Jednotliví démoni si obvykle založí adresář ve /var/log a zaznamenávají svoje hlášky do souborů v něm. Je nutné zmínit, že v jiných distribucích může být (a bývá) nastavení syslogu jiné, a tudíž i obsah jednotlivých souborů či jejich názvy se mohou lišit. Jelikož logy mají tendenci s časem narůstat, provádí se jejich rotace a následný výmaz. Rotace logů způsobí, že po určité definované době se aktuální soubor přejmenuje, popřípadě ještě zkomprimuje, a začne se logovat do nového. Staré logy se pak vymažou. Toto chování má na starosti nástroj logrotate, který má konfigurační soubory v /etc/logrotate.conf a pro jednotlivé služby pak v /etc/logrotate.d. Na závěr ještě doplním, že logování se může nasměrovat nejenom do souborů, ale je také možné jednotlivá hlášení přenášet sítí na nějaký počítač, který bude hlášky skladovat. To pak může pomoci v případě napadení, kdy útočník na napadeném stroji vymaže systémové logy. Pokud ovšem budou všechny hlášky kopírovány nebo ukládány mimo napadený počítač, na kterém bude přístupný třeba pouze démon syslog, pak bude mít útočník smůlu (nebude-li mít k dispozici exploit pro danou verzi syslogu).

### Práce s logy

Možností, jak sledovat či efektivně prohlížet logy je celá řada. Samotné prohlížení logů lze realizovat pohodlně pomocí nástroje less, který patrně dobře znáte, či zless, který slouží k prohlížení komprimovaných souborů. Tyto nástroje jsou interaktivní (na rozdíl třeba od more), po textu je možné se pohybovat a hledat velmi podobným způsobem jako v editoru Vi. Pro úplnost uvádím seznam základních klávesových zkratek:

- h, j, k, l - posun vlevo, dolu, nahoru a vpravo o jeden znak/řádek (ekvivalent šipek)
- g - přesun na začátek souboru
- G - přesun na konec souboru
- / - vyhledat zadaný výraz od kurzoru dolů
- ? - vyhledat zadaný výraz od kurzoru nahoru
- n - posun po směru vyhledávání na další výskyt výrazu
- N - posun proti směru vyhledávání na další výskyt výrazu
- & - zobrazit pouze řádky odpovídající zadanému výrazu
- h - zobrazit nápovědu
- q - ukončení programu

Pokud chcete v logu nebo celém stromu logů hledat nějaký výraz, určitě vám pomůže nástroj grep, popřípadě jeho rekurzivní režim grep -R.

### Sledování logů

Inspekci logů můžete provádět ručně, ale jak už bylo řečeno v úvodu článku, je to přinejmenším nepraktické, zejména, pokud každý den přibudou tisíce či statisíce záznamů. Praktičtější je použít nástroj, který z logů vytáhne to podstatné a pošle vám to e-mailem. Mezi tyto nástroje lze zařadit logcheck alogwatch. Každý z nich pracuje na trošku jiném principu - logcheck posílá e-maily jednou za hodinu a posílá hlášky, které propadly přes všechny jeho filtry, zatímco logwatch (kterým se bude zabývat následující díl seriálu) se snaží určité typy zpráv sumarizovat, a v ideálním případě tak zasílat pouze souhrn událostí, popřípadě statistiky, spíše než konkrétní řádky z logů. V některých případech můžete požadovat ohlášení určité události, jakmile nastane (např. úspěšné přihlášení roota nebo uživatele přes SSH apod.), popřípadě zareagovat na určitou událost jiným způsobem. Takovou funkcionalitu vám může nabídnout nástroj swatch, který se klasifikuje jako aktivní monitorovací nástroj pro logy, tzn. umí na hlášky vámi definovaného typu reagovat vámi nastaveným způsobem.

### Logcheck

Logcheck je jednoduchý, lehký nástroj pro sledování logů. Pro své spouštění využívá cron a vždy jednou za určitou dobu (resp. při každém spuštění z cronu) zašle e-mail s relevantními hláškami z logů (výchozí dobou je jedna hodina). Jeho princip práce spočívá v eliminaci „obvyklých" hlášek které nemají pro správce příliš velký význam, přičemž všechny hlášky, které tyto filtry propustí, se pošlou v těle e-mailu. Cron tabulky náležející jednotlivým démonům naleznete obvykle v /etc/cron.d. V případě Logwatch tomu není jinak, takže jeho cronový zázna naleznete v /etc/cron.d/logcheck, kde si můžete frekvenci jeho spouštění upravit.

#### *Instalace a základní konfigurace*

Nejprve je samozřejmě třeba Logcheck nainstalovat, což lze v Debianu provést následujícím způsobem:

```
aptitude install logcheck
```

Výchozí nastavení v Debianu je plně funkční, ale je vhodné změnit přinejmenším e-mailovou adresu, kam se mají výňatky z logů posílat. Zde postačí upravit hodnotu proměnné SENDMAILTO v hlavním konfiguračním souboru Logchecku, /etc/logcheck/logcheck.conf, podle následujícího vzoru:

```
SENDMAILTO="michal@example.org"
```

Pokud toto nastavení neprovedete, budou se hlášky zasílat na logcheck@localhost, který je aliasem proroot@localhost. To je sice obvykle funkční nastavení, ale vhodnější je posílat výňatky z logů do poštovní schránky mimo server, který sledujete (případný útočník může e-maily z místních schránek vymazat).

#### *Konfigurace Logchecku podrobněji*

Pro úplnost bych rád prošel konfiguraci Logchecku malinko podrobněji. Základem pro jeho nastavení je určení sady pravidel, která se má použít pro filtraci. Tyto sady jsou dostupné celkem tři - wo class="western"rkstation, server a paranoid - a jejich volba ovlivňuje množství filtrovaných informací. Výchozí hodnotou je server, což by mělo pro běžné použití na serveru postačovat. Každý z těchto režimů si samozřejmě budete moci upravit (viz dále). Nastavení konkrétního režimu provedete v proměnnéREPORTLEVEL v /etc/logcheck/logcheck.conf. Zbylé volby jsou velmi dobře okomentované a pro základní funkčnost postačí je ponechat tak, jak jsou. Dalším důležitým souborem je /etc/logcheck/logcheck.logfiles, který, jak naznačuje jeho název, slouží k určení souborů, ze kterých se bude samotný extrakt provádět. V základní konfiguraci Debianu je to syslog aauth.log. Tento seznam můžete samozřejmě podle libosti rozšířit.

#### *Úpravy filtrovacích pravidel*

Sady filtrovacích pravidel se nachází v /etc/logwatch/ignore.d.[režim], kde za [režim] dosaďte hodnotu, kterou jste nastavili v proměnné REPORTLEVEL, tedy ve výchozím stavu by to měl být/etc/logwatch/ignore.d.server. Tento adresář obsahuje značné množství souborů s filtrovacími pravidly. Při filtrování se použijí filtrovací pravidla ze všech souborů v tomto adresáři, takže na názvu souboru nezáleží a není problém si zde vytvořit řadu souborů s vlastními pravidly, kterým filtrování doladíte podle svých představ. Nemusí být od věci svoje vlastní pravidl umisťovat do jasně označených souborů, abyste pak byli schopni dohledat, které soubory pochází z distribuce Logchecku a které jste vytvořili sami. Nutné to ale samozřejmě není. Co se týče pravidel pro filtrování, pak pokud umíte regulární výrazy, nebudete mít sebemenší problém vytvářet vlastní filtrovací pravidla, protože ničeho jiného než regulárních výrazů se zde nepoužívá. Filtrovací pravidlo tedy vypadá třeba takto (tato ukázka pochází ze souboru /etc/logwatch/ignore.d.server/postfix, z distribuce Logchecku):

```
^\w{3} [ :[:digit:]]{11} [._[:alnum:]-]+ postfix/n?qmgr\[[[:digit:]]+\]: [[:alnum:]]+: removed$
```

Tato řádka odstraní hlášky o odstranění e-mailu z fronty Postfixu, tedy obrovskou spoustu hlášek tohoto typu:

```
Mar 28 09:01:12 debian postfix/qmgr[2266]: 841B2214405: removed
Mar 28 09:01:12 debian postfix/qmgr[2266]: 405CF214404: removed
Mar 28 09:10:06 debian postfix/qmgr[2266]: 91427214405: removed
```

```
Mar 28 09:10:06 debian postfix/qmgr[2266]: CEA7D214404: removed
Mar 28 09:10:06 debian postfix/qmgr[2266]: DE3D9214405: removed
Mar 28 09:20:07 debian postfix/qmgr[2266]: 68F4E214405: removed
```

Regulárním výrazům se v tomto článku věnovat nebudu, ale v odkazech pod článkem (nebo v manuálové stránce nástroje grep) naleznete více než dostatek materiálu. Můžete se také podívat podrobněji na existující filtrovací pravidla a podle potřeby je zkopírovat a upravit.

## Správa linuxového serveru: Sledování logů pomocí nástroje logwatch

Minulý díl byl věnován nástroji logcheck, který posílá e-mailem výběr hlášek z logů. Logwatch, kterému se bude věnovat tento díl, se na rozdíl od něj snaží hlášky z logů interpretovat a kompletovat – zasílá tedy spíše přehled důležitých událostí a příslušné statistiky než samotné hlášky.

### Logwatch

Logwatch je patrně nejpoužívanějším nástrojem pro sledování a analýzu logů. Od nástroje logcheckpředstaveného v minulém díle se liší právě tím, že na rozdíl od něj logy analyzuje. V čem je rozdíl? Představte si, že máte zhruba tři stovky hlášek tohoto typu:

```
Apr 08 06:45:40 debian amavis[31840]: (31840-15) Passed CLEAN, LOCAL [127.0.0.1] [127.0.0.1]  -> , Message-ID:
<1302410739.0@example.org>, mail_id: RmtIbaalm5pu, Hits: -4, size: 1200, queued_as: 94208214411, 675 ms
```

Zatímco Logcheck by tyto hlášky buď zahodil, nebo vám je naopak všechny poslal, Logwatch se pokusí tyto hlášky „seskupit" a v zaslaném e-mailu budete mít místo hromady hlášek prostou a jednoduchou informaci, že tolik a tolik e-mailů bylo prověřeno a prošlo:

```
---------- Amavisd-new Begin ----------

253 messages checked and passed.

---------- Amavisd-new End ----------
```

Přehled o úspěšných a neúspěšných přihlášeních přes SSH pak vypadá místo hromady hlášek o zamítnutí či povolení přístupu takto (upozorňuji, že některé identifikátory byly ve výpisu upraveny):

```
---------- SSHD Begin ----------


Illegal users from:
61.132.244.xxx (smtp3.vip.xxx.com): 5 times
74.63.113.xxx (.): 133 times
173.32.89.xxx (CPE002481b244a8-CM0011ae8bba18.cpe.net.cable.xxx.com): 138 times
174.120.122.xxx (mx1.client19.xxx.com.br): 2 times
203.99.60.xxx (mbl-99-60-214.dsl.xxx.pk): 100 times

Users logging in through sshd:
root:
1.2.3.4 (example.org): 8 times
spathi:
4.3.2.1 (example.cz): 65 times


Received disconnect:
11: disconnected by user : 65 Time(s)

SFTP subsystem requests: 61 Time(s)

**Unmatched Entries**
reverse mapping checking getaddrinfo for . [74.63.113.xxx] failed - POSSIBLE BREAK-IN ATTEMPT! : 168 time(s)
reverse mapping checking getaddrinfo for cpe-186-22-34-43.xxx.com.ar [186.22.34.xxx] failed - POSSIBLE BREAK-IN ATTEMPT! : 221 time(s)

---------- SSHD End ----------
```

Ačkoliv to není předmětem tohoto článku, povšimněte si v přehledu klasických automatizovaných útoků na SSH, kde útočníci zkoušejí sadu nejčastěji používaných kombinací jmen a hesel (např. uživatel „test" s heslem „test"). Tento konkrétní server nemá nainstalovaný žádný nástroj pro aktivní obranu a zablokování útočících IP adres, takže jsou vidět poměrně vysoká čísla neúspěšných pokusů o přihlášení. Má však zakázanou autentikaci heslem, takže prakticky veškeré pokusy o přihlášení skončily pro útočníka hláškou „Permission denied (publickey)" bez toho, aby mu vůbec bylo umožněno heslo zadat (útočníci zde očividně použili „hloupý" nástroj, který pro tuto eventualitu nebyl připraven, a tak to i bez ohledu na velmi nízkou možnost dodatečného úspěchu zkoušel znovu a znovu). Ale zpět k nástroji Logwatch.

### Instalace nástroje Logwatch

Prvním krokem, pokud Logwatch ve svém systému ještě nemáte, je jej nainstalovat. V Debianu toho dosáhnete následujícím příkazem:

```
aptitude install logwatch
```

Tento nástroj je natolik rozšířený, že byste ho měli najít prakticky v libovolné distribuci, takže pokud používáte něco jiného než Debian, bude nejspíše stačit podívat se po balíčku stejného jména.

Pokud nechce Logwatch konfigurovat, nemusíte. Po instalaci je již plně funkční a během 24 hodin byste měli očekávat první e-mail, zaslaný na e-mail uživatele root.

### Konfigurace Logwatch

Po instalaci nástroje vám určitě instinkt správce zavelí, abyste se podívali do /etc/logwatch. Tam však naleznete pouze prázdné adresáře bez jakýchkoliv souborů. Dvě další typická umístění pro související soubory s jednotlivými nainstalovanými balíčky jsou v Debianu /usr/share a /usr/share/doc. Výchozí konfiguraci pro Logwatch naleznete v /usr/share/logwatch/default.conf (pozor, to je adresář, ne soubor). Tato konfigurace patří do /etc/logwatch/conf (obojí má stejnou strukturu podadresářů). Můžete se také podívat na dostupnou dokumentaci, kterou najdete v /usr/share/doc/logwatch.

Je jasné, že k samotné analýze logů musí mít Logwatch něco, co mu řekne, jaké hlášky má hledat a jak je má interpretovat. K tomu má Logwatch jednotlivé skripty, napsané v Perlu, které logy parsují a produkují výstup. Pokud by vás zajímaly, můžete je najít v /usr/share/logwatch/scripts/services. Pokud byste si vytvořili vlastní skript, bylo by vhodné jej umístit do /etc/logwatch/scripts.

Mezi dva hlavní konfigurační soubory patří logwatch.conf, který nastavuje samotný Logwatch, aignore.conf, kam patří na jednotlivé řádky regulární výrazy, které by měl Logwatch při svém parsování logů ignorovat. Podobný mechanismus má i Logcheck, který byl představen v minulém díle. I když se v tomto článku regulárním výrazům věnovat nebudu, v odkazech pod článkem naleznete několik zdrojů, kde se je můžete rychle naučit.

Konfigurační soubory, které chcete upravit, si nejprve přesuňte do odpovídajícího umístění v /etc/logwatch, aby vám případný upgrade nástroje nepřepsal vaše pečlivě odladěné nastavení. Hlavní konfigurační soubor, logwatch.conf, je velmi dobře zdokumentován, avšak je poměrně rozsáhlý. Mezi stěžejní parametry je možné zařadit následující:

```
#Formát e-mailu, "text" nebo "html"
Format = text

# E-mailová adresa příjemce
MailTo = root

# From pole zasílaných e-mailů
MailFrom = Logwatch

# Ze kdy se mají záznamy zpracovávat, "all" - všechny, "today" - dnešní, "yesterday" včerejší
Range = yesterday

# Úroveň podrobnosti, "Low" - nízká, "Med" - střední, "High" - vysoká
Detail = Low
```

Jelikož Logwatch neběží jako démon, nýbrž jako perlový skript spouštěný Cronem v určitou dobu, je možné snadno ovlivnit, kdy a jak často budete e-maily dostávat. Toto nastavení naleznete v případě distribuce Debian v souboru /etc/cron.daily/00logwatch.

***Zablokování konkrétního skriptu***

Může se stát, že určité typy informací nechcete dostávat vůbec. V takovém případě se vám může hodit způsob, jak zablokovat určitý skript, který tyto informace generuje. Za tímto účelem se podívejte do hlavního konfiguračního souboru, /etc/logwatch/conf/logwatch.conf. Pokud ho v tomto adresáři nemáte, překopírujte si jej z /usr/share/logwatch/default.conf. V tomto souboru nalezněte následující řádku:

```
Service = All
```

Tato řádka sice povoluje všechny služby, ovšem s výjimkou těch, které deaktivujete později. Pod tuto řádku tedy přidejte řádku se stejnou volbou, avšak minusem před názvem konkrétní služby, kterou chcete zakázat. Kupříkladu, pokud byste takto chtěli zakázat skript „amavis", zapsali byste toto:

```
Service = "-amavis"
```

***Konfigurace konkrétního skriptu***

Jednotlivé skripty mají své konfigurační soubory, a to dokonce dvojího rázu. První je nastavení souborů s logy, ve kterých se budou příslušné záznamy vyhledávat. Toto nastavení by mělo být uloženo do/etc/logwatch/conf/logfiles, přičemž konkrétní konfigurační soubor si můžete překopírovat z výchozí konfigurace v /usr/share/logwatch/default.conf/logfiles a upravit. Jméno souboru odpovídá názvu konfigurovaného skriptu. Skripty však také mohou mít svoje parametry, které upravují jejich činnost, resp. informace, které v e-mailech dostáváte. Pro toto nastavení je vhodné místo v adresáři /etc/logwatch/conf/services, kam příslušný konfigurační soubor skriptu překopírujte z /usr/share/logwatch/default.conf/services. Tyto soubory jsou bohatě zdokumentované, včetně příkladů použití jednotlivých voleb.

**Pár slov na závěr**

Logwatch je nejčastěji používaným nástrojem pro sledování logů, a to z dobrého důvodu. Zasílá zprávy s přehledem událostí spíše než jednotlivé řádky z logů. Nabízí široké možnosti úprav konfigurace i rozšíření jeho záběru. Ale ať už se rozhodnete použít jakýkoliv nástroj, klíčové pro správu jakéhokoliv serveru je logy skutečně sledovat, najít si čas a procházet je. Máte tak šanci odhalit skrytý problém, podezřelou aktivitu či průnik útočníka. Máte-li problém s délkou zpráv, určitě věnujte čas úpravě konfigurace zvoleného nástroje, abyste dostávali zprávy pouze s těmi informacemi, které jsou pro vás zásadní. Určitě není dobře, pokud vám přijde každý den z každého spravovaného serveru 50 kB velký e-mail, kde pak pouze „vyzobáváte" to, co je pro vás skutečně podstatné.

## Správa linuxového serveru: Webový server Cherokee

Nejpoužívanějším webovým serverem je Apache. Existuje však celá řada dalších webových serverů s řadou zajímavých vlastností. Jedním z nich je rychlý a „uživatelsky přívětivý" webový server Cherokee. A právě ten vám představím v tomto dílu.

### Úvod

Cherokee zaujme hned v několika oblastech. V první řadě se jedná o server vyvíjený s velkým důrazem na výkon, měl by tedy být rychlejší než Apache, alespoň ve výchozím nastavení a v určitých situacích (jeden drobný benchmark naleznete v odkazech pod článkem). Webových serverů s důrazem na výkon existuje samozřejmě více, např. Lighttpd či Nginx. Cherokee má ovšem také vlastnost, která je u unixových webových serverů relativně unikátní – webové GUI. Cherokee tedy můžete nastavit pohodlně přes webové administrační rozhraní, alespoň z větší části. Díky tomu je možné uvažovat o Cherokee nejenom pro produkční server, ale také pro testovací stroje vývojářů, i když tam bude Cherokee soutěžit s balíkemXAMPP, který obsahuje Apache, MySQL, PHP a Perl v jedné instalaci.

### Instalace

Samotná instalace webového serveru Cherokee je v Debianu velice jednoduchá, postačí nainstalovat Cherokee z distribučních repozitářů:

```
aptitude install cherokee
```

V případě jiných distribucí by to mělo jít obdobným způsobem, Cherokee je už poměrně zaběhnutý a známý projekt.

### Webové rozhraní

Jak už bylo zmíněno výše, Cherokee se konfiguruje přes webové rozhraní. Konfigurační soubor samozřejmě existuje a sídlí v /etc/cherokee/cherokee.conf. I když je možné jej ručně upravovat, není to doporučeno a samotný soubor neobsahuje prakticky žádné komentáře. Je rozhodně lepší a pohodlnější použít webové rozhraní. Za tímto účelem je třeba administrační webové rozhraní nejprve nahodit, k čemuž slouží následující příkaz:

```
cherokee-admin
```

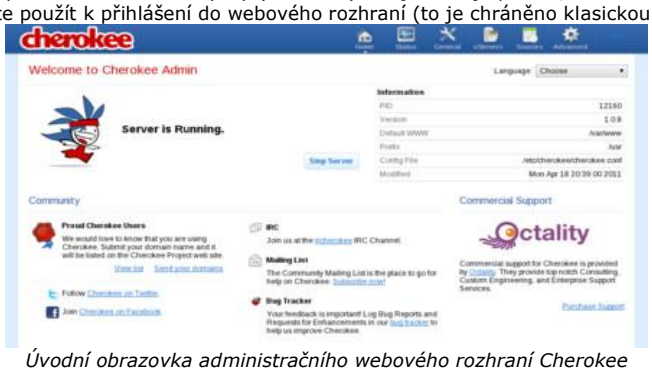
Po spuštění tohoto příkazu se vám objeví výpis podobný tomuto:

```
Login:
User:               admin
One-time Password: ewXpVKjHp2LQbi6v

Web Interface:
URL:                http://127.0.0.1:9090/

Cherokee Web Server 1.0.8 (Jan 12 2011): Listening on port 127.0.0.1:9090, TLS
disabled, IPv6 enabled, using epoll, 4096 fds system limit, max. 2041
connections, caching I/O, single thread
```

Jak je patrné, webové rozhraní běží na portu 9090 místní smyčky (localhost). Stejně tak je patrné, že vám byly přiděleny jednorázové přihlašovací údaje, které můžete použít k přihlášení do webového rozhraní (to je chráněno klasickou HTTP autentikací).




*Úvodní obrazovka administračního webového rozhraní Cherokee*

Pokud vám server běží na localhostu místního počítače, není to problém, stačí spustit prohlížeč a zadat URL http://127.0.0.1:9090/. Běží-li vám server na vzdáleném počítači, máte dvě možnosti. Buď použijete SSH tunel a namapujete si vzdálený port 9090 na místní port, nebo webové rozhraní pustíte na síť, ale v takovém případě by bylo vhodné jej nějakým způsobem ochránit. Zmíněný SSH tunel můžete vytvořit takto:

```
ssh -N -f user@server -L 9090:127.0.0.1:9090
```

Pokud se rozhodnete pro druhou variantu, použijte parametr -b a specifikujte IP adresu, na které mácherokee-admin naslouchat, takto:

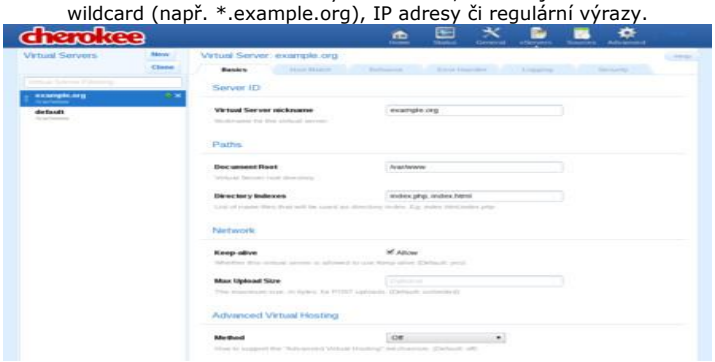
```
cherokee-admin -b 4.3.2.1
```

Před tím by ovšem bylo vhodné daný port zabezpečit, aby se k němu nemohl připojit někdo jiný. Za tímto účelem můžete použít linuxový firewall (Netfilter) a konfigurační nástroj iptables a provést něco podobného tomuto:

```
iptables -I INPUT -p tcp --dport 9090 -j REJECT
iptables -I INPUT -p tcp --dport 9090 -s 1.2.3.4 -j ACCEPT
```

Jen pro informaci, každé z těchto pravidel se vždy přidá na začátek řetězu (parametr -I), proto jsou pravidla zapsána v „obráceném" pořadí. První pravidlo, které skončí jako druhé, zakáže přístup na port 9090. Druhé pravidlo, které skončí jako první, povolí přístup na port 9090 z IP adresy 1.2.3.4.

Samotné konfiguraci se příliš věnovat nebudu, neboť s grafickým (resp. webovým) rozhraním by nastavení mělo být relativně intuitivní. Porty a IP adresy pro naslouchání specifikujete na kartě **General | Ports to listen**. Virtuální servery se konfigurují na kartě **vServers**. Pro každý virtuální server můžete mít vlastní**Document root** (záložka **Basics** u zvoleného virtuálního serveru) a řadu pravidel pro zpracování obsahu (záložka **Behaviour**). To, který virtuální server se vám zobrazí za kterých okolností, nastavujete na kartě**Host match**. Můžete specifikovat jméno, wildcard (např. *.example.org), IP adresy či regulární výrazy.




*Konfigurace virtuálních serverů v Cherokee*

Po jakékoliv změně nezapomeňte konfiguraci uložit klepnutím na **Save** vpravo nahoře. Pokud bude potřeba restart, nabídne vám Cherokee několik možností – nerestartovat, provést bezpečný restart a restartovat natvrdo. Určitě zkuste nejdříve **Graceful restart**.

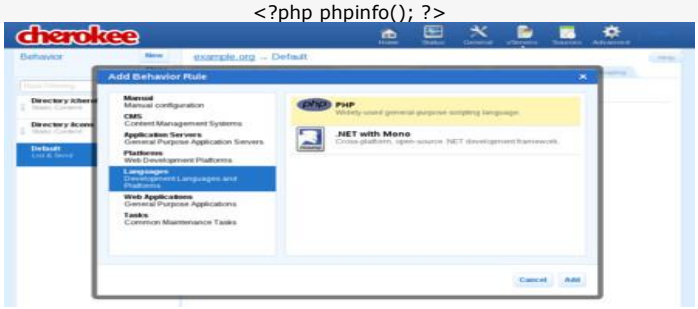
### PHP

Jelikož Cherokee přistupuje k PHP přes rozhraní CGI, je potřeba nainstalovat balíček php5-cgi a ujistit se, že v souboru /etc/php5/cgi/php.ini je řádka s následujícím obsahem:

```
cgi.fix_pathinfo=1
```

Nastavení PHP v Cherokee je uživatelsky přívětivé, jen je důležité vědět, odkud začít. Klepněte na kartu**vServers** a zvolte si požadovaný virtuální server. Podívejte se na kartu **Behaviour** a klikněte na tlačítko**Rule Management**. V levém horním rohu rozhraní vytvořte nové pravidlo klepnutím na **New**. Měl by se spustit průvodce, který vidíte na obrázku níže. Zvolte **Languages** a následně **PHP**. Klepněte na **Add** a následně na **Create**. Konfiguraci nezapomeňte uložit, popřípadě server restartovat. Tím by mělo být nastavení hotové – můžete jej zkusit zkontrolovat vytvořením testovacího souboru s příponou php a obsahem:

```
<?php phpinfo(); ?>
```




*Průvodce pro konfiguraci PHP interpretu v Cherokee*

Dodám, že webové rozhraní Cherokee se poměrně dost měnilo mezi jednotlivými verzemi. Tento návod je určen pro Debian Squeeze a Cherokee 1.0.8. U jiných verzí může být rozvržení webového rozhraní Cherokee mírně odlišné.

## Cherokee a SSL

Konfigurace SSL v Cherokee přívětivá rozhodně není, alespoň v případě Debianu Squeeze. Správce očekávající jednoduché nastavení pak může skončit v e-mailové konferenci Debianu s domněnkou, že podpora pro SSL/TLS nebyla do balíčků Cherokee zakompilována, i když opak je pravdou. Jako první krok nainstalujte knihovnu libssl pro Cherokee. Ta není v závislostech Cherokee, takže nebyla nainstalována společně s ním. To můžete provést následujícím příkazem.

```
aptitude install libcherokee-mod-libssl
```

Poté byste měli nastavit SSL/TLS port. Klepněte na **General | Network**. U položky **SSL/TLS back-end**vyberte z rozbalovacího seznamu možnost **OpenSSL / libssl**. Nyní už můžete nastavit SSL port na kartě**Ports to listen**. Přidejte port **443** a zaškrtněte u něj políčko **TLS/SSL port**.

V tuto chvíli zbývá už jen přiřadit certifikáty jednotlivým virtuálním serverům (karta **Security** u zvoleného virtuálního serveru). Práce s certifikáty již byla v tomto seriálu popsána, konkrétně v článku o konfiguraci Apache. Cherokee implicitně využívá SNI (Server Name Identification), takže můžete hostovat více virtuálních serverů s různými certifikáty na jedné IP adrese. Problém s touto metodou je v (ne)podpoře ze strany některých prohlížečů na některých operačních systémech (viz seznam na Wikipedii). Pokud prohlížeč nepodporuje SNI, Cherokee mu pošle certifikát, který je přiřazen k výchozímu virtuálnímu serveru.

Více informací o nastavení SSL/TLS v Cherokee naleznete na webu projektu.

## Paměťová náročnost a OpenVZ

Z mého testování vyplynula poměrně nepříjemná informace – i když je Cherokee paměťově nenáročný (z hlediska reálně zabrané paměti), rezervuje si velké hodnoty virtuální paměti (neboť spouští poměrně hodně vláken), což na fyzickém stroji nevadí, ale může to vadit u některých virtualizačních řešení, zejména u těch založených na kontejnerech (OpenVZ atd.). Tam se totiž tato paměť, byť nevyužitá, započítává do paměťové kvóty. Nejrychlejším řešením je minimalizovat počet vláken – v konfiguraci Cherokee je to karta**Advanced | Resources** a položka **Thread Number** (nastavte třeba na **1**). V článku na lowendbox.com je navrženo ještě jiné, možná trošku krkolomnější, nicméně také velmi účinné řešení. Dodám, že na Xenu či KVM tento problém není.

## Správa linuxového serveru: Thttpd a benchmark webového serveru

Webový server thttpd je minimalistický, hodí se pro maličké osobní, domácí či embedded servery, popřípadě pro VPS s málo RAM. Jelikož nastavení tohoto serveru je relativně jednoduché, bude se druhá část článku zabývat nástrojem ab pro benchmark webového serveru.

### Úvod

Pro tentokrát začnu nevýhodami. Thttpd je momentálně nevyvíjený, a to již poměrně dlouhou dobu (poslední verze vyšla koncem roku 2003). Mailing list projektu je sice ještě stále aktivní, ale netroufnu si odpovědět na otázku, zdali vývoj tohoto projektu bude ještě někdy pokračovat. Lze však předpokládat, že případné fatální či bezpečnostní problémy budou řešeny jako doposud, tedy na úrovni správců balíčků v distribucích, které thttpd obsahují. Mezi další nevýhody thttpd patří také absence podpory IPv6, HTTPS a FastCGI (podporuje pouze CGI).
Jeho výhody naopak zahrnují paměťovou nenáročnost (sám zabírá pod 1 MB RAM, při zátěži spotřebovává minimum paměti navíc), rychlost (u statických stránek stíhá víc a rychleji než Apache) a vestavěná podpora pro škrcení síťového provozu (throttling).
Jak už bylo zmíněno, thttpd se hodí především tam, kde je citelný nedostatek paměti a kde nevadí absence pokročilejších funkcí typických pro servery typu Apache. Bude vynikat zejména tam, kde je potřeba servírovat statické stránky. Pokud máte produkční server, popřípadě chcete dynamické stránky, budete spíše uvažovat o lighttpd či Nginx.

### Instalace

Je velká pravděpodobnost, že thttpd máte v repozitářích své distribuce, takže by mělo stačit nainstalovat stejnojmenný balíček. V Debianu byste instalaci thttpd provedli takto:

```
aptitude install thttpd
```

### Základní konfigurace

Primární konfigurační soubor pro thttpd je /etc/thttpd/thttpd.conf. Výchozí nastavení je v Debianu postačující pro provoz na portu 80, přičemž obsah, který thttpd servíruje, má výchozí umístění shodné s Apache, tedy /var/www. Pokud vám toto nastavení vyhovuje, můžete přejít rovnou ke spuštění serveru. V opačném případě můžete tyto hodnoty upravit, třeba takto:

```
port=8000
dir=/srv/http
```

Thttpd se na rozdíl od jiných webových serverů v Debianu po instalaci automaticky nespustí, jeho použití je potřeba nejprve povolit v /etc/default/thttpd, kde musíte volbu ENABLED nastavit na yes:

```
ENABLED=yes
```

Teprve poté nastartujte thttpd:

```
/etc/init.d/thttpd start
```

### *chroot a bezpečnost*

Z bezpečnostních důvodů je doporučeno používat chroot, tj. uzavřít běh serveru v adresáři s obsahem, který má thttpd servírovat. Tak je také thttpd v Debianu ve výchozím stavu nastaven. Pokud vám toto nastavení nevyhovuje, zaměňte volbu chroot za nochroot.
Podobně jako Apache nebo jiné webové servery i thttpd po spuštění změní uživatele z roota na neprivilegovaného uživatele, v Debianu je to uživatel www-data (toto upravuje volba user).

### Nastavení throttlingu

Pro domácí servery má thttpd jednu velkou výhodu - má vestavěný throttling, tedy „škrcení" síťového provozu, což se velmi hodí tam, kde máte škrcený upload (mj. prakticky veškeré domácí ADSL). Throttling se nastavuje ve zvláštním konfiguračním souboru, a sice v /etc/thttpd/throttling.conf. Samotné nastavení může vypadat třeba takto:

```
**              5000-10000
**.jpg|**.png   5000
files/**        5000
```

První řádka udává globální nastavení pro všechny soubory, přičemž maximum je stanovené na 10000 bytů za vteřinu. První číslo, tedy 5000, udává minimum. Druhý řádek dává všem jpg a png souborům maximum5000 bytů za sekundu. Poslední řádka naznačuje, jak by se postupovalo pro všechny soubory v určitém adresáři, v tomto případě se jedná o adresář files, kterému je rovněž nastaveno maximum přenosů na5000 bytů za sekundu.

### Benchmark webového serveru pomocí nástroje ab

Přemýšlíte-li nad tím, jakým způsobem změřit výkon webového serveru při zátěži, můžete zkusit nástroj ab z balíčku apache2-utils. K instalaci vám v Debianu postačí následující příkaz:

```
aptitude install apache2-utils
```

Oddělení balíčků je zde velmi prospěšné, neboť nemusíte kvůli tomuto nástroji tahat celou distribuci Apache, postačí vám pouze tento balíček. Nástroj ab má dva klíčové parametry, a sice celkový počet HTTP žádostí (-n) a počet současně posílaných žádostí (-c). Výchozí hodnotou je u obojího jednička. Pokud tedy chcete otestovat místní server zasláním dvou tisíc žádostí, přičemž současně by se posílalo třicet žádostí, vykonali byste následující příkaz:

```
ab -n 2000 -c 30 http://127.0.0.1/
```

Pro ukázku uvádím výstup tohoto příkazu právě v případě thttpd:

```
Server Software:        thttpd/2.25b
Server Hostname:        127.0.0.1
Server Port:            80

Document Path:          /
Document Length:        1399 bytes

Concurrency Level:      30
Time taken for tests:   0.135 seconds
Complete requests:      2000
Failed requests:        0
Write errors:           0
Total transferred:      3272000 bytes
HTML transferred:       2798000 bytes
Requests per second:    14760.58 [#/sec] (mean)
Time per request:       2.032 [ms] (mean)
Time per request:       0.068 [ms] (mean, across all concurrent requests)
Transfer rate:          23582.34 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       2
Processing:     0    2   0.5      2       3
Waiting:        0    1   0.6      1       3
Total:          0    2   0.6      2       4

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      2
  75%      2
  80%      3
  90%      3
  95%      3
```

```
  98%      3
  99%      3
 100%      4 (longest request)
```

Ve výpisu jsou patrné podstatné statistické údaje, tedy nejdelší požadavek, během jaké doby byla obsloužena většina požadavků, přenosová rychlost, počet přenesených bytů apod. Pokud to s bechmarkováním budete myslet vážně, určitě se podívejte do manuálové stránky nástroje kvůli ostatním volbám.

***Pár slov o benchmarkování***

Aby měl jakýkoliv benchmark smysl, je potřeba zadat správné parametry a výsledky správně interpretovat. Už samotná volba testovací stránky skýtá mnoho úskalí. Pokud testujete na statické stránce, ale hodláte nasadit dynamické stránky (resp. webovou aplikaci), nemusí mít test vůbec žádnou vypovídací schopnost. Podobně, testujete-li jednu konkrétní dynamickou stránku, je třeba mít na paměti, že u jiných stránek může být výkon řádově jiný (provádí se jiné SQL dotazy, jiné zpracování dat apod.). Jinými slovy, před započetím testování byste měli mít jasnou představu, co chcete testovat, a měli byste si být jisti, že jste zvolili správnou metodu, jak to otestovat. Určitě nebývá od věci kromě výsledků sledovat i zatížení systému, obsazení paměti a další parametry během samotného benchmarku.

V příkladu výše vás mělo zarazit to, že test byl prováděn z localhostu, tj. přímo na serveru. Takový test může být vhodný, pokud testujete „surový" výkon webového serveru, kde by mohla síťová vrstva výsledek ovlivnit. Naopak pro testování z pohledu uživatelů webu není dobré síť ignorovat, jelikož její režie si vždy něco vezme. V souvislosti s tím byste si také měli dávat pozor, odkud server testujete, aby vám nízký upload vašeho ISP nezkreslil výsledky.

## Správa linuxového serveru: Lighttpd a PHP přes FastCGI

Dalším z webových serverů, který se pyšní menší režií a vyšším výkonem, je lighttpd. Tento díl seriálu se bude věnovat představení lighttpd a zprovoznění PHP v tomto serveru prostřednictvím FastCGI.

### Úvod

Lighttpd má sice na poli webových serverů relativně nízký podíl (zhruba 0,6 % dle Netcraftu), avšak je používán některými velmi známými webovými službami jako např. Wikipedie nebo YouTube. Lighttpd nabízí vysoký výkon, minimální spotřebu paměti a architekturu vhodnou pro AJAXové (resp. Web 2.0) aplikace. Hodí se však nejenom do prostředí náročných webových aplikací s vysokou návštěvností, ale také třeba na osobní, domácí servery či VPS s málo RAM.

### PHP aplikace a webové servery

Řada webových aplikací napsaných (nejen) v PHP je stavěna s ohledem na Apache, což může působit mírné komplikace při nasazení na jiný webový server. Člověk by řekl, že pro aplikaci napsanou v PHP nebo jiném jazyce bude podstatný příslušný interpret a na webovém serveru nebude záležet, ale bohužel tomu tak není. Problém tkví zejména v mechanismech jako rewrite nebo v souborech .htaccess. Řada „alternativních" webových serverů sice rewrite mechanismy podporuje (včetně lighttpd a Nginx), nicméně jejich syntaxe je odlišná a konvertory bývají vzácné (resp. spíše neexistují). To představuje nutnost pravidla ručně přepsat.

Samotné .htaccess soubory, tedy soubory s lokální konfigurací webového serveru specifickou pro daný adresář (a podadresáře) jsou také specifikem Apache a jinde obvykle nejsou podporovány (to je samozřejmě případ i lighttpd). Příslušná pravidla a úpravy konfigurace je tedy třeba zohlednit v samotné konfiguraci webového serveru.

Webové aplikace využívají těchto dvou mechanismů zpravidla ke dvěma účelům. Prvním jsou „čistá" nebo „pěkná" URL (http://www.example.org/index.php?modul=clanky&uzel=354 versushttp://www.example.org/clanky/354) a druhým je ochrana citlivých souborů, u kterých by přímý přístup přes web představoval bezpečnostní riziko. Při nasazení webových aplikací na některé z „alternativních" webových serverů byste měli přinejmenším tuto oblast prozkoumat. U známých a často používaných webových aplikací bývá možné dohledat specifika pro nasazení třeba na lighttpd nebo na jiný webový server, stačí se podívat na web projektu či použít vyhledávač.

### Instalace

Instalace samotného webového serveru je velmi jednoduchá, v Debianu postačí již tradičně použít správce balíčků k nainstalování stejnojmenného balíčku:

```
aptitude install lighttpd
```

### Konfigurace lighttpd

Pokud si ještě vzpomínáte na Úvod do konfigurace Apache, Debian si konfiguraci webových serverů, zejména tedy Apache, přizpůsobuje a strukturuje, aby bylo nastavení a jeho změny co nejjednodušší. V případě lighttpd se to promítá pouze do správy modulů, které je možné „zapínat" a „vypínat" prostřednictvím nástrojů lighttpd-enable-mod a lighttpd-disable-mod. Ty pak vytvářejí a ruší symlinky z/etc/lighttpd/conf-available, kde se nacházejí dostupné konfigurační soubory jednotlivých modulů, do/etc/lighttpd/conf-enabled, kde se nacházejí konfigurační soubory, které bude lighttpd brát v úvahu a načte z nich příslušné nastavení.

Hlavním konfiguračním souborem je zde /etc/lighttpd/lighttpd.conf, který je poměrně strohý a obsahuje jen základní nastavení. Určitě si zde přizpůsobte některé z následujících voleb:

- server.document-root – klasický document root, kořenový adresář webové prezentace
- server.dir-listing – umožní procházet obsah adresáře, pokud mu chybí indexový soubor
- index-file.names – všechny názvy souborů, které se berou jako indexové
- server.port - port, na kterém má lighttpd naslouchat

#### *PHP*

Na rozdíl od Apache nemá lighttpd vlastní „konektor" pro obsluhu PHP aplikací, nicméně to není problém – můžete použít FastCGI. Za tímto účelem je ovšem potřeba do systému dostat CGI interpret PHP z balíčkuphp5-cgi, pokud jej ještě nemáte:

```
aptitude install php5-cgi
```

Poté je potřeba upravit konfiguraci PHP pro CGI interpret tak, aby byla povolena volba cgi.fix_pathinfo. Upravte tedy soubor /etc/php5/cgi/php.ini a odstraňte středník (uvozující komentář) z řádky s touto volbou. Všimněte si, že konfigurace PHP pro CGI interpret je oddělena od konfigurace PHP pro Apache nebo pro jiná užití, třeba pro PHP interpret pro příkazovou řádku.

Poté je třeba povolit příslušné moduly, konkrétně modul fastcgi a fastcgi-php. Můžete ručně vytvořit příslušné symbolické odkazy, nebo použít nástroje pro obsluhu modulů zmíněné výše:

```
lighttpd-enable-mod fastcgi
lighttpd-enable-mod fastcgi-php
```

Nyní už zbývá jen donutit lighttpd znovu načíst konfiguraci:

```
/etc/init.d/lighttpd force-reload
```

Otestovat funkci PHP interpretu můžete vytvořením a pokusem o zobrazení libovolného souboru s příponou.php s obsahem (zobrazit by se měl výpis konfigurace PHP interpretu):

```
<?php phpinfo(); ?>
```

Pokud chcete nasadit nějakou webovou aplikaci, budete nejspíše potřebovat ještě nějaký databázový server. Instalace databázového serveru byla již probrána v článcích Instalace LAMP (MySQL) aPostgreSQL: Instalace a zprovoznění. Jediné, co je potřeba zajistit kromě instalace databázového serveru, je propojení databáze s PHP interpretem, tedy dostupnost PHP knihoven pro práci s příslušnou databází (balíčky php5-mysql či php5-pgsql).

#### *SSL*

Chcete-li zprovoznit přístup k webovému serveru přes HTTPS, čtěte dále. Samotná problematika HTTPS a SSL včetně generování self-signed certifikátů či získání certifikátu zdarma od některé z volných certifikačních autorit byla v tomto seriálu již probrána. Budu tedy předpokládat, že máte k dispozici certifikát s klíčem.

Pokud máte certifikát a klíč ve dvou souborech, spojte je do jediného .pem souboru (určitě nastavte jeho práva tak, aby jej nemohl číst nikdo jiný než webový server!):

```
cat certifikat.crt klic.key > certifikat.pem
```

Nyní povolte ssl modul lighttpd:

```
lighttpd-enable-mod ssl
```

Ještě před tím, než donutíte webový server znovu načíst konfiguraci, upravte cestu k .pem souboru v/etc/lighttpd/conf-enabled/10-ssl.conf tak, aby odpovídala skutečnému umístění souboru s certifikátem a klíčem. Teprve poté proveďte znovunačtení konfigurace a pokus o připojení přes HTTPS:

```
/etc/init.d/lighttpd force-reload
```

#### *Rewrite*

Jak už bylo řečeno výše, rewrite pravidla pro Apache nebo jiné servery mají jinou syntax než pravidla prolighttpd, tudíž je třeba je přizpůsobit. Samotnou dokumentaci k modulu pro rewrite lighttpd naleznete nawiki projektu. Dodám jen, že modul pro rewrite není ve výchozí konfiguraci aktivní, musíte jej povolit v/etc/lighttpd/lighttpd.conf, konkrétně ve volbě server.modules odkomentováním řádky obsahující"mod_rewrite".

## Správa linuxového serveru: Webový server Nginx

Nginx je rostoucí hvězdou mezi alternativami k nejpoužívanějšímu webovému serveru Apache. Stejně jako lighttpd nebo Cherokee patří i Nginx k lehkým, nenáročným, ale přesto velice výkonným serverům. V tomto díle vám jej přiblížím.

### Úvod

Projekt Nginx pochází z Ruska, napsal jej Igor Sysoev a je dostupný pod velmi liberální BSD licencí. Nginx je dle statistik v současné době druhým nejpopulárnějším linuxovým webovým serverem (pokud počítáte webové servery obecně, pak je třetí, hned za IIS od Microsoftu). Pokud jste si prošli minulý díl, pak prakticky všechny základní charakteristiky sdílí s webovým serverem lighttpd, tj. je rychlý, výkonný a má minimální spotřebu paměti. Výborně se tedy hodí jak pro provoz náročných aplikací, tak pro provoz na menších serverech či osobních VPS.

Je schopen fungovat také jako reverzní proxy, tedy předávat požadavky jinému webserveru a vracet klientovi odpověď, čímž se výborně se hodí pro rozložení zátěže (load balancing) náročné aplikace na více fyzických (či virtuálních) webových serverů. Zde je třeba dodat, že projektů pro load balancing existuje více, a sice reverzní proxy Pound či HAProxy (toto jsou skutečně pouze reverzní proxy, nejsou schopny zastávat funkci webového serveru a nabízet nějaký obsah samy o sobě). Funkcionalitu reverzní proxy obsahují ovšem i jiné webové servery, např. Apache (v modulu mod_proxy) a lighttpd (opět modul proxy).

Zajímavostí je, že Nginx je schopen pracovat i jako POP3/IMAP poštovní proxy s patřičnou podporou SSL a TLS.

### Instalace

Nginx je běžnou součástí linuxových distribucí, takže stačí najít v repozitářích balíček s odpovídajícím názvem. V Debianu postačí k instalaci použít následující příkaz:

```
aptitude install nginx
```

### Základní konfigurace

V Debianu je konfigurace Nginx opět lehce přizpůsobená. Jádrem je konfigurační soubor/etc/nginx/nginx.conf, avšak nastavení je rozděleno ještě do dvou dalších adresářů. Nastavení v souborech v /etc/nginx/conf.d budou načtena při startu Nginx (resp. bude naincludována hlavním konfiguračním souborem). Konfiguraci jednotlivých virtuálních serverů (virtual host) můžete rozdělit do souborů v /etc/nginx/sites-available, přičemž aktivní virtuální servery je pak třeba symlinkovat do/etc/nginx/sites-enabled. Je to podobné jako u Apache, jen zatím chybí shellové nástroje pro snadnější aktivaci a deaktivaci jednotlivých serverů, takže si budete muset vystačit s ručním vytvářením symbolických odkazů. Příslušné skripty budou k dispozici patrně až pro další vydání Debianu, které přijde po Debian Squeeze (pokud používáte Debian unstable, což na server nelze doporučit, měli byste mít tyto nástroje v balíčku s Nginx už teď).

Z hlediska konfigurace je třeba brát v úvahu, že Nginx je primárně reverzní proxy, takže HTTP server se v konfiguraci bere jako samostatný modul (dostupný v sekci http). Sekce jsou v tomto případě bloky uvozené klíčovým slovem a oddělené pomocí složených závorek:

```
http {

volby

}
```

Jednotlivé sekce mohou mít vnořené podsekce. V případě HTTP serveru je hierarchie následující: http ->server -> location.

Sekcí server a location může být přirozeně více.

Základní chování Nginx se nastavuje mimo sekci http. Definují jej zejména následující volby (tento úsek pochází z výchozí konfigurace Debianu):

```
user www-data;
worker_processes  1;

error_log  /var/log/nginx/error.log;
pid        /var/run/nginx.pid;

events {
worker_connections  1024;
# multi_accept on;
}
```

Volba user nastavuje uživatele, pod kterým budou Nginx procesy běžet. Podstatný je počet procesů (worker_processes), který je vhodné nastavit podle počtu dostupných jader procesoru. V úseku výše můžete také vidět sekci modulu events, kde se skrývá podstatná volba worker_connections, která určuje, kolik klientů může najednou obsloužit jeden proces. Celkový počet najednou obsluhovaných klientů je tedy roven součinu worker_processes a worker_connections.

Samotné nastavení sekcí server (která odpovídá virtuálnímu serveru) a location (která definuje jednotlivá umístění či typy souborů a jejich vlastnosti a přístupová práva) je poměrně názorně naznačeno v/etc/nginx/sites-enabled/default.

Základní konfigurace virtuální serveru může vypadat třeba takto:

```
server {
listen 1.2.3.4:80;
server_name www.domain.com;
location / {
root   /var/www;
}
}
```

Nastavení IP adresy a portu pro naslouchání by mělo být jasné, stejně jako jméno serveru. Všimněte si, že kořenový adresář (document root v Apachi) se nastavuje pomocí location.

Pokročilejší konfigurace je velmi dobře popsána na wiki Nginx (viz odkazy v závěru článku), včetně příkladů pro virtuální servery, load balancing i použití SSL (to je už dokonce v konfiguraci výchozího virtuálního serveru v Debianu připraveno, tedy pouze zakomentováno) apod.

### *Moduly a Nginx*

Nginx má k dispozici řadu modulů rozšiřující jeho funkcionalitu, avšak na rozdíl od Apache není možné moduly snadno přidávat nebo odebírat - moduly totiž musejí být zvoleny při kompilaci (jsou pak zakompilovány do příslušné binárky). Pokud byste tedy chtěli nějaký modul, který není zakompilován v binárce, kterou dodává vaše distribuce, není jiná možnost než Nginx překompilovat ručně.

### Nginx a PHP

Nginx přistupuje k PHP stejně jako lighttpd, tedy přes FastCGI. Na rozdíl od Lighttpd však Nginx neobsahuje nástroj pro spuštění FastCGI procesů, musíte si tedy vypomoci nástrojem spawn-fcgi:

```
aptitude install spawn-fcgi
```

Stejně jako v případě Lighttpd je třeba upravit nastavení PHP interpretu odkomentováním řádkycgi.fix_pathinfo=1 v souboru /etc/php5/cgi/php.ini.

Následně je třeba spustit FastCGI procesy, které bude moci Nginx využít k běhu PHP aplikací. To je bohužel nutné udělat ručně:

```
spawn-fcgi -a 127.0.0.1 -p 9000 -u www-data -g www-data -f /usr/bin/php5-cgi -P /var/run/fastcgi-php.pid
```

Abyste zajistili spuštění FastCGI procesů po startu, můžete tuto řádku přidat do /etc/rc.local, což zajistí provedení příkazu po každém spuštění systému. Jistou výhodou tohoto přístupu je mj. možnost změnit konfiguraci PHP bez restartu webového serveru - postačí pouze restart FastCGI procesů.

Posledním krokem je editace příslušného virtuálního serveru (např. v souboru /etc/nginx/sites-available/default), kde vložíte nebo odkomentujete a upravíte následující část:

```
location ~ \.php$ {
fastcgi_pass   127.0.0.1:9000;
fastcgi_index  index.php;
fastcgi_param  SCRIPT_FILENAME  /var/www$fastcgi_script_name;
}
```

Zde je potřeba upravit zejména řádku s fastcgi_param, kde místo /var/www uveďte cestu ke kořenu webové prezentace (document root) daného virtuálního serveru.

V konfiguraci je potřeba přizpůsobit ještě jednu věc, a sice přidat index.php jako jeden z možných indexových souborů:

```
location / {
    root   /var/www;
    index  index.php index.html index.htm;
}
```

Tím by mělo být nastavení PHP v Nginx hotovo, stačí jej už jen otestovat a začít používat.

**Který server vybrat?**

V předchozích dílech byly představeny prakticky všechny nejpoužívanější webové servery, tj. Apache, Nginx, Lighttpd a Cherokee. Každý z těchto serverů má svoje přednosti a charakteristiky. Nemyslím si, že existuje obecná „kuchařka", jak vybrat ten správný server, neboť výběr silně závisí na vlastnostech, které potřebujete, dále pak na tom, co na daném serveru chcete nasadit, jaký hardware máte k dispozici, a hlavně, jak velkou zátěž bude váš server generovat.

Nakonec vám asi nezbude než si jednotlivé servery otestovat právě ve vašem prostředí a v případě projektů, které chcete nasadit, a na základě výsledků testování se pak rozhodnete. Pokud budete testovat s využitím jakéhokoliv benchmarku (např. nástroje ab, který byl představen v tomto díle), určitě neberte v úvahu pouze naměřené hodnoty, ale také zátěž CPU a obsazení paměti během testu. Může se totiž klidně stát, že server, který bude o pár milisekund pomalejší, bude zatěžovat CPU či paměť třeba jen z poloviny oproti svému konkurentovi.

Jak už jsem nakousl v minulém díle, řada webových aplikací je stavěna s ohledem na provoz na nejrozšířenějším webovém serveru, tedy na Apachi. „Kompatibilita" aplikace s jinými servery se pak odvíjí od toho, jak moc (a zdali vůbec) aplikace využívá soubory .htaccess a pravidla pro rewrite URL. To jsou dvě hlavní oblasti rozdílů mezi webovými servery, .htaccess je specifikum Apache, nicméně pravidla pro řízení přístupu v něm obsažená lze snadno přidat do konfigurace jiného webového serveru, jen bude potřeba zohlednit rozdíly v syntaxi. Přepis URL je o něco horší, neboť tam je syntax o něco složitější, ale ani to není neřešitelný úkol. U řady projektů je možné použít vyhledávač a s trochou štěstí se dostat přímo k pravidlům, která již někdo vámi zvolenému serveru přizpůsobil.

## Správa linuxového serveru: SNI, webové servery a prohlížeče

SNI (Server Name Indication) je metoda umožňující provoz více HTTPS virtuálních webů (virtual host) s různými certifikáty na jedné jediné IP adrese. V tomto článku se dozvíte, jak to vypadá s podporou SNI na straně prohlížečů i na straně serverů, a pochopitelně se také dozvíte, jak SNI nakonfigurovat na nejpoužívanějších linuxových webových serverech.

### Úvod

Zásadním problémem HTTPS bylo po dlouhou dobu pořadí vykonávaných činností, kdy se v prvním kroku vytvořilo SSL spojení, a teprve přes něj se přenesl HTTP požadavek se jménem webu, který klient požadoval. To znamenalo, že webový server nemohl při vytváření SSL spojení vědět, který certifikát má klientovi předložit. V praxi to znamenalo (a bohužel stále v případě mnoha serverů znamená), že na jedné IP adrese mohly být virtuální weby pouze s jedním certifikátem. Pokud jste chtěli použít pro každý virtuální web jiný certifikát, museli jste si zajistit odpovídající počet IP adres a každé IP adrese přiřadit konkrétní certifikát (a webovou prezentaci).

Tento problém se bohužel nedá dost dobře řešit na úrovni certifikátů – i když je možné vytvořit certifikát pro více domén, certifikát může být vydán pouze pro jednu osobu či organizaci. Řešení tedy muselo přijít v podobě rozšíření protokolů SSL a TLS, a právě tomuto rozšíření se říká SNI (Server Name Indication). Toto rozšíření umožňuje prohlížeči předat jméno požadovaného webu ještě před vytvářením šifrovaného spojení, což dává serveru možnost servírovat i na jediné IP adrese různé certifikáty podle toho, který web klient požaduje.

### Podpora v prohlížečích

Aby mohlo SNI fungovat, musí být podporováno na straně serveru, ale také na straně klienta. Jako správci serverů máte serverovou část pod kontrolou, tudíž hlavní problém spočívá v podpoře na straně klientů. V této oblasti se sice hnuly ledy a řada prohlížečů dnes SNI bez problémů podporuje, ale ani zdaleka ne všechny a ne na všech systémech. Problémy jsou zejména na Windows XP a mobilních zařízeních, ale i na Linuxu. Aktuální seznam naleznete na Wikipedii. Dle tohoto zdroje SNI podporují následující verze prohlížečů:

- IE 7 a výše, ovšem ne na Windows XP, kde SNI v IE nefunguje bez ohledu na verzi
- Mozilla Firefox 2.0 a vyšší
- Opera 8.0 a vyšší
- Google Chrome (na XP až od verze 6)
- Safari 2.1 a výše
- Android – výchozí prohlížeč od verze Honeycomb
- atd.

SNI naopak nepodporuje třeba Konqueror, IE a Safari na XP, wget, Windows Mobile až po 6.5, Oracle Java JSSE atd. Co se Konqueroru týče, podpora SNI snad velmi brzy přibude (měla by být součástí verze 4.7), ale než se nový Konqueror dostane do distribucí, chvíli to ještě potrvá. Jak je vidět, není to sice úplně špatné, ale stejně tak to ještě pořád není to pravé ořechové. Pokud se tedy budete rozhodovat o nasazení SNI, určitě berte na vědomí, že ne všichni vaši klienti jej patrně podporují.

S tím se pojí zásadní otázka – co se stane, pokud váš server navštíví klient nepodporující SNI? Server by měl zareagovat více či méně podle očekávání, tzn. měl by použít výchozí certifikát (obvykle první certifikát v definici virtuálních webů) a po vytvoření SSL spojení naservírovat klientovi správný web. Klient se tedy na web dostane, ale ne bez obvyklého varování o problému s předloženým certifikátem.

### Podpora ve webových serverech

Dle Wikipedie podporují SNI webové servery v následujících verzích:

- Apache 2.2.12 a výše
- Cherokee zkompilované s podporou TLS
- lighttpd 1.4 a 1.5 s patchem nebo 1.4.24 a výše bez patche
- Nginx 0.5.32 a vyšší, podpora musí být zakompilována a podmínkou je OpenSSL podporující SNI, tj. verze 0.9.8f a vyšší

S výjimkou serveru Apache, kde zmiňuji i řešení pro starší systémy (Debian Lenny), by příklady uvedené níže měly fungovat až v Debianu Squeeze s tím, že na starších verzích Debianu jsem testování neprováděl.

#### *SNI a Apache*

Máte-li k dispozici Apache 2.2.12 nebo novější, můžete použít mod_ssl. Na starších systémech musíte použít mod_gnutls. Následující ukázka je určena pro mod_ssl:

```
Listen 443

NameVirtualHost *:443

<VirtualHost *:443>
DocumentRoot /var/www/example1.cz
ServerName www.example1.cz

SSLEngine On
SSLCertificateFile /etc/apache2/example1.pem
SSLCertificateKeyFile /etc/apache2/example1.key

</VirtualHost>

<VirtualHost *:443>
DocumentRoot /var/www/example2.cz
ServerName www.example2.cz

SSLEngine On
SSLCertificateFile /etc/apache2/example2.pem
SSLCertificateKeyFile /etc/apache2/example2.key

</VirtualHost>
```

Jak je vidět, na samotném nastavení není nic obtížného, postačí klasický NameVirtualHost a pak jednotlivé virtuální weby lišící se jménem a certifikáty. V případě, že Apache získá požadavek od nekompatibilního klienta, použije certifikát prvního takto definovaného virtuálního webu, tedy v tomto případě to budeexample1.

Pokud byste raději, aby Apache v případě nekompatibilního klienta vygeneroval chybovou hlášku 403 a nedovolil klientovi přístup, nastavte volbu SSLStrictSNIVHostCheck na „on" (výchozí nastavení je „off", takže pokud tuto volbu neuvedete, Apache se zachová, jak bylo uvedeno výše). Máte-li starší systém se starší verzí Apache, zbývá vám kromě možnosti použít novější Apache z backportů jen mod_gnutls. Konfigurace SNI tímto způsobem by vypadala takto (nezapomeňte modul gnutls nejprve nainstalovat a aktivovat):

```
Listen 443

NameVirtualHost *:443

<VirtualHost *:443>
DocumentRoot /var/www/example1.cz
ServerName www.example1.cz

GnuTLSEnable on
```

```
GnuTLSExportCertificates on
GnuTLSCertificateFile /etc/apache2/example1.pem
GnuTLSKeyFile /etc/apache2/example1.key

</VirtualHost>

<VirtualHost *:443>
DocumentRoot /var/www/example2.cz
ServerName www.example2.cz

GnuTLSEnable on
GnuTLSExportCertificates on
GnuTLSCertificateFile /etc/apache2/example2.pem
GnuTLSKeyFile /etc/apache2/example2.key

</VirtualHost>
```

***SNI a Lighttpd***

Pokud máte aktivovaný modul ssl, postačí do definice virtuálních webů přidat direktivu ssl.pemfile s odkazem na soubor obsahující jak certifikát, tak privátní klíč, takto:

```
$HTTP["host"] == "www.example1.cz" {
server.document-root = "/var/www/example1.cz"
ssl.pemfile = "/etc/lighttpd/server.pem"
}
$HTTP["host"] == "www.example2.cz" {
server.document-root = "/var/www/example2.cz"
ssl.pemfile = "/etc/lighttpd/server2.pem"
}
```

***SNI a Cherokee***

Cherokee používá SNI již ve výchozím nastavení jako mechanismus pro odlišení SSL virtuálních webů, takže zde není návod třeba. Jediné, na co byste si měli dát pozor, je přiřazení certifikátu výchozímu virtuálnímu serveru, jehož certifikát se použije v případě, že klient SNI nepodporuje.

***SNI a Nginx***

Použití SNI v případě serveru Nginx je také relativně jednoduché, resp. postačí podobná strategie jako v případě serveru Apache, tj. normálně definovat virtuální weby běžící na portu 443 se zapnutým ssl (příklad viz níže). Stejně jako u Apache platí i zde, že certifikát prvního virtuálního webu se použije jako výchozí certifikát pro nekompatibilní klienty.

Abyste si ověřili, zdali Nginx podporuje SNI, resp. byl zkompilován s podporou SNI, můžete použít následující příkaz:

```
nginx -V
```

Ve výpisu byste měli vidět následující řádku:

```
TLS SNI support enabled
```

Pokud ji neuvidíte, nezbude vám než si Nginx zkompilovat ručně. Následuje příklad konfigurace SNI v Nginx:

```
server {
listen   443;
server_name  www.example1.cz;

ssl  on;
ssl_certificate  /etc/nginx/example2.pem;
ssl_certificate_key /etc/nginx/example2.key;

location / {
root   /var/www/example1.cz;
index  index.html index.htm;
}
}

server {
listen   443;
server_name  www.example2.cz;

ssl  on;
ssl_certificate  /etc/nginx/example1.pem;
ssl_certificate_key /etc/nginx/example1.key;

location / {
root   /var/www/example2.cz;
index  index.html index.htm;
}
}
```

**Pár slov na závěr**

V úvodu zmíněný problém s HTTPS řeší nejenom SNI, ale také svým způsobem protokol IPv6, i když přechod klientů na IPv6 bude možná podstatně pomalejší než přechod klientů na prohlížeče schopné používat SNI. Faktem nicméně je, že podpora SNI na straně klientů v tuto chvíli stále není bezproblémová a nějakou dobu ještě nebude, což celkem logicky brzdí nasazení SNI na produkční servery.

## Správa linuxového serveru: XMPP (Jabber) server Prosody

Instant messaging, tedy komunikace prostřednictvím „okamžitých zpráv", se stala velmi populární. Existuje celá řada protokolů, služeb i klientů sloužící k realizaci tohoto typu komunikace. Jedním z protokolů je i Jabber/XMPP, který je populární nejenom ve sféře uživatelů Linuxu a svobodného softwaru. Jakým způsobem je možné postavit si vlastní XMPP server, je předmětem tohoto dílu.

### Obecně o Jabberu a XMPP

Jak už bylo řečeno, existuje řada protokolů pro výměnu „okamžitých zpráv". Řada z nich je však uzavřená, centralizovaná a podléhá restriktivním licenčním podmínkám či ještě navíc nutnosti instalovat proprietární software, který komunikaci zajišťuje (v některých případech nemusí být takový software pro Linux ani dostupný). XMPP je protokol, který je decentralizovaný a otevřený. Jednou z jeho stěžejních vlastností je i rozšiřitelnost, která umožňuje na Jabberu stavět další služby a činit jej tak pro uživatele atraktivnější.

Otevřenost protokolu a související možnost jej použít volně, dle libosti, dala vzniknout řadě implementací XMPP serverů i klientů. Oproti řadě jiných protokolů a podobných služeb má Jabber ještě jednu zásadní odlišnost – je decentralizovaný. To znamená, že nemá žádný centrální bod, na kterém by komunikační síť závisela a bez kterého by nikdo s nikým nemohl komunikovat. Jabber se z tohoto pohledu podobá e-mailu - serverů je celá řada, mohou spolu navzájem komunikovat, přičemž uživatelé se mohou rozhodnout, jaký server si zvolí, popřípadě mají možnost si založit vlastní Jabber server (viz dále), a stále mají možnost komunikovat s uživateli ostatních serverů.

Jabber nebo XMPP? Dříve se protokol XMPP jmenoval Jabber. Nyní se toto označení používá spíše pro službu, zatímco novější XMPP se používá pro samotný protokol. Vztah mezi výrazy Jabber a XMPP je tak podobný jako mezi výrazy e-mail a SMTP.

Uživatelé Jabberu jsou identifikováni pomocí JID (Jabber ID), které má totožný tvar jako e-mailová adresa:jmeno@domena.cz.

### Klienty

Klientů pro Jabber existuje ohromné množství. Na tomto serveru byly mj. představeny multiplatformní klienty Psi, Coccinella a Gajim. Známým klientem je i multiplatformní a multiprotokolový Pidgin nebo rovněž multiprotokolový klient pro prostředí KDE, Kopete. Existují také klienty pro konzoli určené pro milovníky příkazové řádky, jedná se např. o klient Mcabber či s Pidginem distribuovaný Finch. Jsou rovněž dostupné klienty pro webový prohlížeč i mobilní zařízení. Podrobnější seznam klientů naleznete jednak na Wikipedii, na wiki českého Jabber portálu a také na webu XMPP nadace.

### Servery

Jabber serverů existuje také celá řada. Existují v C napsané servery jabber14 a jabber2, v Erlangu napsaný ejabberd, v Javě napsané Openfire a Tigase a v neposlední řadě i odlehčený server Prosody napsaný v Lua (ten bude představen v tomto dílu). Tento soupis rovněž není úplný, obsáhlý seznam jsem nalezl nawebu XMPP nadace.

### Prosody

Prosody je lehký XMPP server napsaný v jazyce Lua a vydaný pod MIT/X11 licencí. Snaží se o jednoduchost konfigurace (to budete moci posoudit níže) a nenáročnost na zdroje (to mohu potvrdit, jelikož mi běží na dvou serverech a na obou si alokuje kolem 10 MB virtuální paměti). Díky tomu se výborně hodí na osobní VPS nebo servery s omezenými zdroji.

#### *Instalace*

Ve vámi používané distribuci byste mohli najít balíček Prosody přímo v repozitářích (pokud ne, nezbude vám nic jiného než Prosody nainstalovat ze zdrojových kódů). Distribuce založené na Debianu by měly mít Prosody v repozitářích k dispozici, nicméně vývojářský tým tohoto serveru nabízí ještě svůj vlastní repozitář, kde byste měli najít vždy aktuální verzi. Tak či onak, pro instalaci by měl postačit příkaz:

```
aptitude install prosody
```

#### *Konfigurace*

Pokud sledujete tento seriál pravidelně, určitě tušíte, že po vzoru konfigurace webových serverů bude distribuce Debian strukturu konfiguračních souborů pro Prosody upravovat podobným stylem. Konfiguraci naleznete v adresáři /etc/prosody. Hlavním konfiguračním souborem je prosody.cfg.lua, pro konfigurační soubory jednotlivých virtuálních hostů je určen adresář conf.avail, přičemž aktivní konfigurační soubory je třeba nalinkovat symbolickým odkazem do conf.d.

Jako první asi budete chtít vytvořit nastavení pro vámi používanou doménu, tedy vytvořit příslušný virtual host, který umístíte do /etc/prosody/conf.avail a vytvoříte symbolický odkaz na něj do/etc/prosody/conf.d. Nejjednodušší záznam vypadá takto:

```
VirtualHost "example.cz"
```

Nastavení v prosody.cfg.lua jsou nastavení globální. Nastavení pod jednotlivými direktivami VirtualHostby měla být specifická pro daný virtual host. Můžete tak specifikovat odlišná nastavení pro různé domény, zejména pak nastavení jednotlivých komponent, např. modulu pro konference (kde pak můžete specifikovat konkrétní doménu):

```
Component "conference.example.cz" "muc"
```

Výchozí nastavení v Debianu má zakázané registrace uživatelů a jedinou funkční doménou je localhost. Po nastavení jedné nebo více domén můžete uvažovat o úpravě hlavního konfiguračního souboru a přizpůsobit si globální nastavení. Mezi podstatné volby patří:

- specifikace správců: `admins = { "admin@example.cz", "admin@example.net" }`
- seznam aktivovaných modulů: `modules_enabled`
- povolení či zakázání registrace uživatelů: `allow_registration`
- uvítací zpráva zaslaná právě registrovaným uživatelům: `welcome_message`
- seznam uživatelů, kterým budou zasílány informace o registracích uživatelů: `registration_watchers`

Obecně lze doporučit projít si tento soubor a jeho komentáře a upravit si nastavení podle svých představ.

#### *Vytváření a rušení uživatelských účtů*

Prosody má k dispozici nástroj pro řízení jeho chodu, a sice prosodyctl. Tímto nástrojem můžete provádět změny za běhu, zejména pak přidání uživatele, odebrání uživatele a změna hesla uživatele:

- přidání uživatele: prosodyctl adduser uzivatel@domena.cz
- odebrání uživatele: prosodyctl deluser uzivatel@domena.cz
- změna hesla uživatele: prosodyctl passwd uzivatel@domena.cz

#### *SRV záznamy DNS*

Preferovaným způsobem navazování spojení v rámci XMPP klientů i serverů je využívání SRV DNS záznamů, které udávají, kde konkrétně běží příslušné služby (viz příslušné RFC). V případě XMPP se jedná o dvě služby, resp. dva porty, port pro připojení klientů (5222) a port určený pro komunikaci mezi servery 5269. Z tohoto hlediska je velmi vhodné pro XMPP server příslušné SRV záznamy nastavit. Tyto záznamy mají následující podobu:

```
_xmpp-server._tcp.example.cz.  IN SRV   0 0 5269   hostname.
_xmpp-client._tcp.example.cz.  IN SRV   0 0 5222   hostname.
```

Zde nahraďte example.cz za název domény, kde chcete nasadit Jabber server (toto je doménové jméno v JID); hostname pak nahraďte za jméno serveru, kde váš Jabber server běží. Pokud chcete, můžete upravit i čísla portů. Dvě nuly zde udávají prioritu a váhu (v tomto pořadí).

Následující SRV záznam sice již ze specifikace XMPP protokolu zmizel a jakýkoliv software pracující s XMPP aktualizovaný během posledních několika let by jej neměl vyžadovat, nicméně z důvodu zpětné kompatibility může být vhodné zvážit přidání i tohoto záznamu. Přinejmenším je to jedna z věcí, kterou byste měli vést v patrnosti při řešení případných problémů, pokud tento SRV záznam nepoužijete.

```
_jabber._tcp.example.cz.       IN SRV   0 0 5269   hostname.
```

#### *SSL/TLS nastavení*

Jabber komunikaci je velmi vhodné šifrovat, nejlépe platným certifikátem, kterému budou vaši klienti věřit. Problematika certifikátů a možnosti získání důvěryhodného certifikátu zdarma byly probrány dříve v seriálu. Zde už předpokládám, že certifikát k dispozici máte. Specifikace certifikátu by měla přijít do nastavení příslušného virtual hostu, konkrétně do sekce ssl:

```
VirtualHost "example.cz"

        ssl = {
                key = "/etc/certs/example.key";
                certificate = "/etc/certs/example.crt";
```

```
}
```

Podstatné volby ovlivňující nastavení SSL/TLS jsou především tyto:

- `c2s_require_encryption` - komunikace klienta se serverem (c2s) vyžaduje šifrování
- s2s_require_encryption - komunikace mezi servery (s2s) vyžaduje šifrování

Více o pokročilém nastavení SSL/TLS v Prosody se dozvíte v dokumentaci.

**Jabber a transporty**

Na závěr zmíním možnost komunikace s ostatními sítěmi pomocí transportů. Chcete-li propojit svůj Jabber/XMPP server se sítěmi jako ICQ, AIM, MSN apod., můžete použít některý z projektů jako např.spectrum. Na jeho wiki naleznete prakticky veškeré potřebné informace, jak tuto komponentu nastavit a integrovat s Prosody. Spectrum není součástí Debianu Squeeze, ale projekt má k dispozici repozitáře pro Debian, tudíž použitím tohoto nástroje nepřijdete o snadnou aktualizaci.

## Správa linuxového serveru: Jabber (XMPP) server ejabberd

V minulém díle jsem nakousl problematiku Jabber/XMPP serverů. Tento díl volně navazuje na předchozí a představí vám robustní a výkonný XMPP server ejabberd.

### Úvod

Pokud jste si neprošli minulý díl seriálu, doporučuji, abyste tak učinili, jelikož se v něm nachází některé obecné informace o Jabberu/XMPP, zejména pak nastavení SRV záznamů v DNS.

Ejabberd je výkonný XMPP server napsaný v jazyce Erlang a šířený pod licencí GNU/GPL. Pokud se minule představený server Prosody výborně hodil na soukromé VPS a na servery pro relativně málo uživatelů, ejabberd se výborně hodí na větší servery s větším provozem, kde se také velmi často používá. To mu ovšem přesto nebrání fungovat i na menších serverech, kde se dá rovněž nasadit (pouze spotřeba paměti v porovnání s Prosody je o něco vyšší). Ejabberd je používán mnoha známými subjekty, jako např. BBC Radio, KDE Talk, největším českým Jabber serverem Jabbim či jistou velmi známou sociální sítí.

Ejabberd podporuje fungování v režimu clusteru, replikaci i rozložení zátěže, je tedy pomocí něj možné postavit řešení s vysokou dostupností. Provoz více domén (virtual hosting) je rovněž samozřejmostí. Jako datové úložiště je používán distribuovaný SŘBD (DBMS) Mnesia, který byl vyvinut pro jazyk Erlang. Prostřednictvím ODBC je však možné použít i jiné SŘBD jako např. MySQL či PostgreSQL.

Jelikož Mnesia je distribuovaná databáze, ukládá si kvůli zachování konzistence i jméno příslušného uzlu, tedy hostname. Změna hostname nebo přesun databáze na jiný stroj má pak za následek odmítnutí spuštění ejabberdu. Chcete-li změnit hostname nebo přesunout databázi jinam, podívejte se na heslo „Change Computer Hostname" v dokumentaci projektu, kde je postup popsán. Aby to nebylo jednoduché, tento postup funguje pouze v ejabberdu řady 2.1.x (to je ta, která je momentálně v Debianu Squeeze), pro řadu 2.0.x (Debian Lenny) je k dispozici jiný návod, který je o něco složitější a také trochu dobrodružnější, jelikož se v něm nevyhnete použití Erlang shellu.

### Instalace

Čtete-li tento seriál pravidelně, tušíte už asi, že instalace bude obnášet instalaci stejnojmenného balíčku. V Debianu by to šlo provést příkazem:

```
aptitude install ejabberd
```

Jak je v Debianu zvykem, součástí instalace balíčku je i jeho zprovoznění, ejabberd se tedy po instalaci rovnou spustí.

### Základní konfigurace

Pokud s ejabberdem začínáte, není od věci jako první krok zálohovat konfigurační soubor (`/etc/ejabberd/ejabberd.cfg`). Stejně jako v případě XMPP serveru Prosody, konfigurační soubor je de facto psán v jazyce, ve kterém je napsán daný server. Syntax je tedy podstatně odlišná od toho, co znáte z klasických unixových konfiguračních souborů, a není těžké udělat chybu, kvůli které se pak server už nespustí. Proto je dobré mít zálohu, ke které se můžete vrátit nebo se kterou můžete porovnat aktuální stav.

#### *Obsluhované domény (virtual hosty)*

Jako první zvolte doménu nebo domény, které bude ejabberd obsluhovat. K tomu slouží proměnná hosts, přičemž výchozí hodnotou je „localhost". Tuto hodnotu tedy změňte na příslušnou doménu, takto:

```
{hosts, ["example.cz"]}.
```

Specifikovat můžete samozřejmě i více domén, takto:

```
{hosts, ["example.cz", "example.org"]}.
```

#### *Nastavení certifikátu pro jednu doménu*

Součástí instalace ejabberd je vygenerovaný certifikát /etc/ejabberd/ejabberd.pem, podepsaný sám sebou. Obsluhujete-li jen jednu doménu a máte k dispozici vlastní certifikát, upravte cestu v proměnné certfile v rámci specifikace portu 5222 (předposlední řádek výpisu):

```
{5222, ejabberd_c2s, [
      {access, c2s},
      {shaper, c2s_shaper},
      {max_stanza_size, 65536},
      %%zlib,
      starttls, {certfile, "/etc/ejabberd/ejabberd.pem"}
      ]},
```

#### *Nastavení certifikátů pro více domén*

Máte-li více obsluhovaných domén a více certifikátů, je změna malinko složitější. Jako první krok odstraňte specifikaci certifikátu (`certfile`) z výše uvedeného výpisu, takto:

```
{5222, ejabberd_c2s, [
      {access, c2s},
      {shaper, c2s_shaper},
      {max_stanza_size, 65536},
      %%zlib,
      starttls
      ]},
```

Poté na globální úroveň, třeba těsně za konec sekce listen (tj. za první výskyt znaků „]}." bez uvozovek od počátku definice sekce listen) umístěte definice doménových certifikátů:

```
{domain_certfile, "example.cz", "/etc/ejabberd/example_cz.pem"}.
{domain_certfile, "example.org", "/etc/ejabberd/example_org.pem"}.
```

#### *Aktivace starého SSL portu 5223*

Tento port by se v dnešní době již neměl používat, nicméně kvůli starým klientům můžete zvážit jeho aktivaci. Je zakomentovaný, stačí jej tedy odkomentovat:

```
{5223, ejabberd_c2s, [
      {access, c2s},
      {shaper, c2s_shaper},
      {max_stanza_size, 65536},
      zlib,
      tls, {certfile, "/etc/ejabberd/ejabberd.pem"}
      ]},
```

#### *Správa uživatelů z příkazové řádky*

Pomocí nástroje `ejabberdctl` můžete ejabberd řídit z příkazové řádky. Relevantní příkazy pro správu uživatelských účtů jsou:

- vytvoření uživatele: `ejabberdctl register jméno doména.cz heslo`
- smazání uživatele: `ejabberdctl unregister jméno doména.cz heslo`
- změna hesla uživatele: `ejabberdctl change_password jméno doména novéHeslo`

#### *Webové rozhraní a nastavení účtu správce*

Ejabberd obsahuje jednoduché webové rozhraní pro správu a zobrazení statistik. Toto rozhraní běží na portu 5280 a vyžaduje HTTP autentikaci. Jelikož ve výchozím stavu není nastaven žádný správce, nemůžete se k němu nijak přihlásit. Za tímto účelem si vytvořte nějaký Jabber účet:

```
ejabberdctl register jméno doména.cz heslo
```

Poté upravte nastavení administrátora v konfiguračním souboru, takto:

```
{acl, admin, {user, "jméno", "doména.cz"}}.
```

Následně server restartujte. Pak byste měli mít možnost se přihlásit k webovému rozhraní prostřednictvím následujícího URL: http://doména.cz:5280/admin.

### Zabezpečení

Ejabberd je trošku problematičtější zabezpečit, jelikož otevírá více portů nad rámec standardních 5222 a 5269, které slouží k připojení klientů a komunikaci mezi servery (v tomto pořadí). Tyto porty samozřejmě není třeba zavírat, nicméně port 5280, kterému náleží webové rozhraní ejabberdu, světu vystavovat nemusíte. Zde můžete buď webové rozhraní omezit firewallem, nebo jej zrušit úplně. Úplné zrušení provedete zakomentováním definice portu 5280, takto:

```
%%  {5280, ejabberd_http, [
%%                %%{request_handlers,
%%                %% [
%%                %%  {["pub", "archive"], mod_http_fileserver}
%%                %% ]},
%%                %%captcha,
%%                http_bind,
%%                http_poll,
%%                web_admin
%%                ]}
```

Zde upozorním na jeden problém – tato položka je na konci seznamu, tudíž za specifikací portu 5280 již není čárka (viz konec posledního řádku výpisu). Za poslední položkou seznamu nesmí čárka následovat, tudíž ji musíte odstranit z poslední nezakomentované položky seznamu (jelikož ta se právě stala poslední, a tudíž má na konci nepovolenou čárku). Pokud jste neprováděli žádné změny sekce listen a používáte Debian Squeeze, pak poslední nezakomentovanou položkou před specifikací portu 5280 je specifikace portu 5269. Na jejím konci uvidíte „]},“, a zde stačí odstranit čárku.

Kromě portu 5280 je velmi vhodné, ne-li ještě vhodnější, zablokovat port 4369, který náleží démonu epmd (Erlang Port Mapper Daemon). To je malý nameserver, který je využíván Erlang aplikacemi při komunikaci mezi uzly v clusteru. K němu potřebuje mít přístup ejabberd běžící na místním stroji (můžete tedy zakázat přístup odjinud než z lokální smyčky). Stavíte-li cluster, musí mít k tomuto portu přístup všechny jeho uzly.

**Aktualizace ejabberdu**

Při povýšení verze vámi používané distribuce si na ejabberd určitě dejte pozor, jelikož se občas stane, že přechod na novou verzi není hladký a vyžaduje ruční zásah. Určitě si přečtěte poznámky k novému vydání a k procesu aktualizace, nebo ještě lépe – upgrade si někde bokem vyzkoušejte.

## Správa linuxového serveru: Konfigurace XMPP serveru ejabberd

Minulý díl se věnoval zprovoznění serveru ejabberd a naprosto základnímu nastavení. Tento díl bude v konfiguraci ejabberdu pokračovat, zmíní propojení ejabberdu s jinou než vestavěnou databází, nastavení traffic shapingu a registrace uživatelů.

### Použití MySQL a PostgreSQL s ejabberdem

Jak už tušíte z předchozího dílu seriálu, ejabberd využívá jednoduchý databázový systém Mnesia napsaný pro jazyk Erlang (v tom je ejabberd napsán). Pokud chcete, je možné využít i jiný databázový systém, a to jak MySQL a PostgreSQL, tak i jiné databáze (v tomto článku budou popsány pouze tyto dvě).

Zprovoznění ovšem není úplně triviální, alespoň ne v Debianu, neboť zde chybí moduly pro propojení ejabberdu s MySQL či PostgreSQL.

#### *Krok 1: Kompilace modulů pro propojení s databází*

Jelikož je třeba provádět kompilaci, potřebujete mít k dispozici nástroje pro její provedení. V Debianu lze vše potřebné jednoduše nainstalovat pomocí metabalíku build-essential, který v závislostech přitáhne kompilátory i ostatní nezbytné nástroje. Kromě těchto nástrojů je třeba mít k dispozici hlavičkové soubory Erlangu (balík erlang-dev) a příslušné nástroje Erlangu (balík erlang-nox se svými závislostmi). A jelikož budete samotné moduly stahovat ze SVN repozitáře, potřebujete ještě navíc Subversion:

```
aptitude install build-essential erlang-dev erlang-nox subversion
```

Poté je třeba stáhnout, zkompilovat a umístit příslušné moduly. Postup se liší dle použité databáze. Nejprve stáhněte ejabberd moduly ze SVN. Z bezpečnostního hlediska doporučuji jak stažení, tak kompilaci provádět s právy neprivilegovaného uživatele:

```
svn co https://svn.process-one.net/ejabberd-modules
```

V závislosti na vámi zvolené databázi zvolte správný adresář se zdrojovými kódy:

```
cd ejabberd-modules/mysql/trunk/
cd ejabberd-modules/pgsql/trunk/
```

Pak spusťte skript, který potřebné moduly sestaví:

```
./build.sh
```

Posledním krokem je umístění modulů do správného adresáře. Pokud jste výše uvedený postup realizovali s právy uživatele, nyní je potřeba získat oprávnění roota:

```
cp ebin/* /usr/lib/ejabberd/ebin/
```

Instalace potřebných modulů je nyní hotová. Zřejmou nevýhodou je v tomto případě umístění souborů mimo balíčkovací systém. Ideální by bylo vytvořit deb balíček pro dané moduly a instalovat jej pomocí dpkg, ale to už je mimo záběr tohoto dílu.

#### *Krok 2: Vytvoření databáze*

Tvorba databáze i uživatele je závislá na typu databázového systému, následují tedy instrukce pro jednotlivé DBMS.

MySQL

Nejprve je třeba získat administrátorská oprávnění databáze, tedy přihlásit se jako `root`:

```
mysql -u root -p
```

Druhým krokem je vytvoření databáze a uživatele pomocí následujících SQL příkazů:

```
CREATE DATABASE jmeno_databaze DEFAULT CHARACTER SET utf8 COLLATE utf8_czech_ci;
GRANT ALL PRIVILEGES ON jmeno_databaze.* TO 'uzivatel'@'localhost' IDENTIFIED BY 'heslo';
```

Poté je třeba v databázi vytvořit příslušné schéma, tj. naplnit ji tabulkami. Toto schéma je k dispozici v adresáři /usr/share/doc/ejabberd/examples. Následující sada příkazů jej zkopíruje do aktuálního adresáře, dekomprimuje a poté importuje příslušné schéma (po poskytnutí správného hesla):

```
cp /usr/share/doc/ejabberd/examples/mysql.sql.gz .
gzip -d mysql.sql.gz
mysql -u uzivatel -p jmeno_databaze < mysql.sql
```

PostgreSQL

Získání oprávnění databázového správce a spuštění konzole s jeho právy lze v případě Postgresu provést takto:

```
su postgres -
psql
```

Následují SQL příkazy k vytvoření uživatele a databáze:

```
CREATE ROLE uzivatel WITH LOGIN PASSWORD 'heslo';
CREATE DATABASE "jmeno_databaze" OWNER "uzivatel" TEMPLATE template0 LC_COLLATE 'cs_CZ.UTF8' LC_CTYPE 'cs_CZ.UTF8';
```

Posledním krokem je import databázového schématu (poslední příkaz předpokládá běžící Postgres server na localhostu):

```
cp /usr/share/doc/ejabberd/examples/pg.sql.gz .
gzip -d pg.sql.gz
psql -W -h localhost -U uzivatel jmeno_databaze < pg.sql
```

#### *Krok 3: Úprava konfigurace ejabberdu*

Nyní je třeba adekvátně upravit /etc/ejabberd/ejabberd.conf. Ještě předtím, než začnete s úpravami, je vhodné tento soubor zazálohovat. Jelikož je to přímo zdroják v Erlangu, je dobré mít funkční verzi bokem pro případ, že byste někde udělali nějakou chybu.

V první řadě je třeba zakomentovat řádku využívající interní identifikační metodu (řádek zakomentujete tak, že na začátek řádku umístíte dva znaky procenta):

```
%% {auth_method, internal}.
```

Pak odkomentujte řádku určující odbc jako autentikační metodu:

```
{auth_method, odbc}.
```

V tuto chvíli, dle použité databáze, odkomentujte a nastavte řádku s přihlašovacími údaji do databáze:

```
{odbc_server, {mysql, "127.0.0.1", "jmeno_databaze", "uzivatel", "heslo"}}.
{odbc_server, {pgsql, "127.0.0.1", "jmeno_databaze", "uzivatel", "heslo"}}.
```

U následujících položek je třeba přidat k jejich názvu přídomek _odbc, tedy např. z mod_roster udělatmod_roster_odbc. Tím zajistíte zpracování dat příslušného modulu databází. Seznam modulů následuje:

- `mod_last` (datum, kdy odpojený uživatel naposledy přistupoval k serveru)
- `mod_offline` (uložené zprávy pro nepřipojené uživatele)
- `mod_private` (uložená soukromá XML data, např. konfigurace klientů atd.)
- `mod_privacy` (pravidla soukromí uživatelů)
- `mod_pubsub` (veřejné služby)
- `mod_roster` (seznamy kontaktů)
- `mod_vcard` (profily uživatelů)

Poté restartujte ejabberd a zkuste, jestli běží:

```
ejabberdctl status
```

Pokud ne, podívejte se do logů ve /var/log/ejabberd, tam by se měla objevit případná chybová hláška, od které se pak můžete odrazit. Většinu chyb způsobí asi chyby syntaxe, tudíž by mohlo pomoci porovnat zálohu (kterou jsem doporučoval vytvořit) s aktuální verzí. K tomu lze použít nástroj diff:

```
diff -u /etc/ejabberd/ejabberd.cfg ejabberd_zaloha.cfg
```

Pokud se ejabberd spustí, můžete vyzkoušet vytvořit prvního uživatele, zdali se to povede (můžete se i poté přesvědčit náhledem do databáze, zdali se vytvořil příslušný záznam uživatele v tabulce users, čímž ověříte, že ejabberd používá váš DBMS místo vestavěného DBMS Erlangu – Mnesia):

```
ejabberdctl register jmeno_uzivatele domena.cz heslo
```

## Traffic shaping

V rámci síťového provozu ejabberdu je možné nastavovat omezení toku (shapery). V současné době je k dispozici jediný typ shaperu – maxrate, který omezuje maximální rychlost průtoku dat na určitý počet bytů za sekundu. Zvýší-li se průměrný průtok nad tuto hodnotu, přeruší se čtení ze socketu na tak dlouho, dokud se průměrná rychlost nesníží pod tuto hodnotu.

Ve výchozí konfiguraci jsou definovány dva shapery, normal a fast:

```
{shaper, normal, {maxrate, 1000}}.
{shaper, fast, {maxrate, 50000}}.
```

Jejich rychlosti (v bytech za sekundu) si můžete upravit podle libosti, popřípadě přidat další shapery. Jejich použití je pak zakotveno v sekci access rules:

```
%% For all users except admins used "normal" shaper
{access, c2s_shaper, [{none, admin},
                      {normal, all}]}.

%% For all S2S connections used "fast" shaper
{access, s2s_shaper, [{fast, all}]}.
```

Shaper normal je automaticky přiřazen všem uživatelům s výjimkou správců. Komunikace mezi servery (S2S) má nastavený shaper fast.

## Veřejný Jabber server: Povolení registrace uživatelů

Ve výchozí konfiguraci není povolena registrace uživatelů, takže registrovat uživatele lze ručně, způsobem naznačeným o kousek výše nebo v minulém díle seriálu. Chcete-li postavit veřejný Jabber server, budete patrně chtít registraci uživatelů povolit. Za tímto účelem nalezněte následující řádku s pravidlem přístupu:

```
{access, register, [{deny, all}]}.
```

V ní změňte deny na allow. To k samotnému povolení registrace stačí, ale je ještě několik věcí, kterým byste měli věnovat pozornost. V první řadě je to omezení frekvence registrací na jednu IP adresu, aby vám nějaký zlý robot najednou nevytvořil kvanta uživatelských účtů. Ejabberd umožňuje omezit frekvenci registrace pomocí ochranné lhůty, tj. po provedené registraci jsou další pokusy zablokovány na určitou dobu. Tuto dobu (v sekundách) můžete nastavit následujícím parametrem:

```
{registration_timeout, 600}.
```

Výchozí hodnota je nastavena na šest set sekund, tedy deset minut. K úplnému odstranění ochranné lhůty použijte hodnotu infinity.

Samotné nastavení mod_register obsahuje dvě podstatné položky. První je welcome_message, tedy uvítací zpráva, kterou uživatel dostane po registraci. Chcete-li ji upravit, najděte a upravte položkuwelcome_message:

```
{welcome_message, {"Welcome!",
                   "Welcome to a Jabber service powered by Debian. "
                   "For information about Jabber visit "
                   "http://www.jabber.org"}},
```

A konečně za poslední, a sice možnost specifikovat účet, kterému budou chodit zprávy o registraci všech uživatelů:

```
{registration_watchers, ["admin@example.cz"]},
```

## Správa linuxového serveru: Rootkity

Když se útočníkovi podaří proniknout na váš server a získat patřičná práva, obvykle nainstaluje tzv. rootkit, tedy sadu nástrojů pro snadnější ovládnutí kompromitovaného počítače. Co přesně rootkity jsou a jak se jim bránit, to se dozvíte v tomto dílu seriálu.

### Úvod

Rootkit je sada nástrojů zajišťující několik věcí. Nejčastěji se v souvislosti s nimi zmiňuje skrývání přítomnosti útočníka (a rootkitu samotného) v systému. Děje se tak obvykle modifikací standardních nástrojů jako ps, ls, netstat tak, aby neukazovaly soubory, procesy a síťová spojení patřící rootkitu nebo jeho „uživateli".

Funkce rootkitu ale také obvykle zahrnují vytvoření zadních vrátek, tedy mechanismu, kterým může útočník získat do kompromitovaného systému pozdější přístup. Rookit také může provádět různé „šmírování", tedy např. zaznamenávání přístupových údajů uživatelů, když se přihlašují, zaznamenávání stisků kláves atd. Kromě toho může obsahovat i nástroje na zametání stop, mazání logů a realizaci nejrůznějších typů útoků.

Jiný druh rootkitů používá místo modifikace systémových nástrojů LKM, tedy modulů jádra, které lze za běhu systému zavést. Po zavedení příslušného modulu může rootkit skrýt svou přítomnost efektivněji, modifikací systémových volání (syscall). Tím si zajistí, že bez ohledu na použitý nástroj nebudou komponenty rootkitu vidět.

Rootkity nejsou pouze doménou unixových systémů – existují i v systémech od společnosti Microsoft. Takové rootkity byly dokonce jednou využity společností Sony BMG jako „ochrana" proti kopírování hudebních CD. Postiženo bylo dle Wikipedie přes sto titulů. Aby toho ještě nebylo málo, časem vyšlo najevo, že součástí „ochrany" proti kopírování bylo i využití softwaru chráněného GNU/GPL a GNU/LGPL licencí, který byl samozřejmě využit v rozporu s licencí. Některé laptopy patrně sužuje nebo sužoval rootkit v BIOSu, který uměl zpřístupnit NTFS oddíl a nainstalovat se do Windows po spuštění počítače. Oficiálním důvodem byla údajně snaha o možnost vystopování ukradených notebooků, bohužel za cenu kompromitace systému a možnosti jedné firmy vzdáleně ovládat kvanta počítačů (ano, tomu se obvykle říká botnet). Ale to jen tak na okraj. Jak je vidět, rootkity se mohou dostat do vašeho systému nejrůznějšími cestami, nikoliv nezbytně ze sítě. Mohou být součástí instalovaného proprietárního softwaru, ale mohou být klidně i součástí svobodného softwaru, který někdo poupraví.

### Běžná prevence

Do běžné prevence patří všechna základní pravidla zabezpečení serverů, tedy především znalost konfigurace daného serveru, všeho, co na něm běží, přehled o dění na něm (čtení logů). Sledování dění kolem Linuxu, zejména pak kolem vámi používané distribuce, sledování bezpečnostních hlášení. Pravidelná aktualizace, resp. alespoň brzká aplikace bezpečnostních záplat, dále izolace běžících služeb - buď pomocí oprávnění, nebo pomocí chrootu či virtualizace. Řízení přístupu k souborům a službám, použití firewallu. Silná hesla, omezený přístup k SSH (pouze z vnitřní sítě, VPN či z konkrétní IP adresy). Vhodné zacházení s uživateli, poskytování minima práv nutných k jejich činnosti. Pravidelné zálohování a testování záloh.

Důležité je mít pod kontrolou veškerý software, který na server instalujete. Riziko obecně představuje software stažený mimo distribuční repozitáře (které, alespoň v případě Debianu, mají všechny balíčky podepsané, takže není jednoduché je podvrhnout). Nedávno podobný problém postihl FTP server vsftpd, kam útočník propašoval zadní vrátka. Kdo si důsledně kontroloval GPG podpisy, popřípadě kontrolní součty, přišel na to. Rozhodně tedy software mimo distribuční repozitáře podrobte dostatečné kontrole, kontrolujte podpisy i kontrolní součty.

Napadení distribučních repozitářů je velmi málo pravděpodobné, i když ne *úplně* nemožné. Obvykle však něco podobného nelze přehlédnout, pokud alespoň trochu sledujete aktuality kolem Linuxu či vaší distribuce, jelikož se kolem toho nepochybně strhne patřičná bouře.

### Vzdálené logování

Jelikož většinou logy sledujete pomocí nástrojů jako Logwatch nebo Logcheck, které zasílají zprávy jednou za delší dobu, není pro útočníka problém případné stopy v logu smazat hned po průniku (příslušný rootkit mu v tom může dokonce pomoci). Tomuto může zabránit technika zvaná vzdálené logování, tedy zasílání kopií všech zpráv na patřičně zabezpečený vzdálený server, a to okamžitě. V takovém případě, i když útočník smaže hlášky z napadeného serveru, hlášky uložené na vzdáleném, logovacím serveru, zůstanou k dispozici.

V případě syslogd (balíček sysklogd) nastavíte vzdálené logování následujícím způsobem. Na serveru (počítači, který bude přijímat hlášky ze vzdálených systémů) je potřeba povolit příjem zpráv v souboru/etc/default/syslogd, takto (změna se projeví po restartu logovacího démona):

```
SYSLOGD="-r"
```

Síťová komunikace probíhá na UDP portu 514, lze tedy doporučit vytvořit příslušná pravidla firewallu k omezení přístupu jen na ty počítače, ze kterých chcete přijímat logované hlášky.

Poté je třeba nastavit klienty. V případě těch je třeba upravit hlavní konfigurační soubor syslogu, tedy/etc/syslog.conf. Zde postačí místo cílového souboru pro uložení logovaných hlášek zapsat zavináč následovaný jménem nebo IP adresou serveru, takto:

```
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
mail,news.none          @server.example.org
```

Po restartu logovacích démonů na klientech by mělo vzdálené logování fungovat a logy na serveru by se měly začít plnit hláškami z klientů. Jelikož hostname bývá součástí hlášek, není problém od sebe odlišit jak jednotlivé klienty, tak hlášky samotného serveru.

### Nástroje pro sledování změn v souborech

Existují nástroje, které sledují změny v kritických souborech, a umožňují tak případnou kompromitaci systému detekovat. Mezi zástupce patří např. Tripwire nebo Aide.

### Detektory rootkitů

Detektory rootkitů jsou specializované nástroje, které využívají nejrůznější techniky pro zjištění přítomnosti rootkitů v systému. Umí zjistit přítomnost známých rootkitů, ale hledají kromě stop konkrétních rootkitů i obecné stopy, tudíž teoreticky dokáží odhalit i přítomnost některých neznámých rootkitů. Je vhodné je používat spolu s nástroji pro sledování změn v souborech (viz výše). Mezi zástupce patří chkrootkit arkhunter.

### Závěr

Rootkity představují jedno z možných rizik, nikoliv však jediné. I když se útočníkovi nepodaří získat roota, stále má možnost nainstalovat nástroje sloužící k pozdějšímu snadnému přístupu (pomocí uživatelské cron tabulky zajistí jejich spuštění po restartu serveru), stejně jako nástroje určené pro útoky na jiné servery. Často se k tomu využívá IRC bot, který se přihlásí do nějaké chatovací místnosti, kde pak útočník vhodným příkazem zajistí provedení příslušných akcí (DDoS útok atd.). Jediná „dobrá" zpráva v tomto případě je, že bez práva roota jsou síťová spojení i tyto procesy viditelné minimálně všemi ostatními uživateli.

V příštím díle představím podrobněji detektory rootkitů a osvětlím obecný postup, co dělat v případě kompromitace systému.

## Správa linuxového serveru: Detekce rootkitů a zotavení

V minulém díle jsem probral základní otázky týkající se rootkitů, včetně obecných pravidel zabezpečení, jak minimalizovat riziko průniku útočníka do systému. Představil jsem také vzdálené logování a ukázal, jak jej nastavit. V tomto díle představím detektory rootkitů a obecné rady, jak postupovat po případném průniku.

### Detektory rootkitů

Detektory rootkitů jsou nástroje určené k vyhledávání jednak známých rootkitů, ale také jejich typických příznaků – mohou tedy (ale také nemusí) detekovat i dosud neobjevený a nepopsaný typ rootkitu. Je ovšem také třeba mít na paměti, že v případě kompromitace systému nelze věřit vůbec ničemu, včetně nainstalovaných detektorů rootkitů. Není tedy vhodné předpokládat, že pokud detektor neohlásí průnik, znamená to s jistotou, že v systému opravdu žádný rootkit není. Na stranu druhou, pokud detektor ohlásí nějaký problém, neznamená to nutně, že ke kompromitaci systému skutečně došlo. Samy detektory jsou bohužel velmi známé tím, že generují falešné poplachy. Kupříkladu, na jednom z mých serverů tvrdí chkrootkit toto:

```
Checking `bindshell'...  INFECTED (PORTS:  465)
```

Hádáte správně, na portu 465 mi běží Postfix (číslo portu odpovídá SSL variantě protokolu SMTP). Jelikož stejný port využívá i bindshell; chkrootkit považuje tento otevřený port za známku infekce, a generuje v tomto případě falešný poplach. Každý poplach, který detektory nahlásí, je vhodné prověřit, a teprve na základě vlastního úsudku (v kombinaci s použitím vyhledávače) rozhodnout, jedná-li se o planý poplach nebo o reálný průnik.

### chkrootkit

Jedním z detektorů rootkitů je chkrootkit. Nainstalujte jej obvyklým způsobem z repozitáře své distribuce, v Debianu a jeho derivátech to zařídí příkaz:

```
aptitude install chkrootkit
```

K provedení kontroly postačí pustit samotný nástroj bez parametrů:

```
chkrootkit
```

Bude následovat výpis výsledků kontroly, velmi pravděpodobně i s nějakým poplachem. Výpis analyzujte a prověřte. Doporučuji si projít informace o falešných varováních v/usr/share/doc/chkrootkit/README.FALSE-POSITIVES. Až se s nástrojem a jeho výpisy seznámíte, můžete si u opakovaných spouštění ušetřit čas využitím tichého režimu, který hlásí pouze nalezené problémy:

```
chkrootkit -q
```

Spolu s tímto nástrojem je nainstalován i skript, který zajišťuje jeho spuštění cronem (/etc/cron.daily/chkrootkit). Ten ovšem ve výchozím nastavení není aktivní. Abyste jej aktivovali, musíte upravit jeho konfigurační soubor (/etc/chkrootkit.conf). Jelikož je opravdu maličký, dovolím si ho sem umístit celý i s výchozími volbami:

```
RUN_DAILY="false"
RUN_DAILY_OPTS="-q"
DIFF_MODE="false"
```

Aby se skript spouštěl a kontroloval přítomnost rootkitů každý den, je třeba proměnnou RUN_DAILY nastavit na hodnotu true. Jelikož je veliká šance, že narazíte na nějaký falešný poplach, doporučuji prozkoumat i volbu DIFF_MODE, která po korektním (dodatečném) nastavení zajistí, že výsledek již provedené kontroly bude vždy porovnán s aktuálním stavem a chkrootkit vás bude informovat pouze o změnách vůči uloženému stavu. Chcete-li této možnosti využít, nastavte nejprve proměnnou DIFF_MODE na hodnotu true. Poté skript ručně spusťte:

```
/etc/cron.daily/chkrootkit
```

V mém případě obsahuje výstup následující:

```
ERROR: No file /var/log/chkrootkit/log.expected
This file should contain expected output from chkrootkit

Today's run produced the following output:
--- [ BEGIN: cat /var/log/chkrootkit/log.today  ] ---

/usr/lib/pymodules/python2.5/.path /usr/lib/pymodules/python2.6/.path /lib/init/rw/.ramfs

INFECTED (PORTS:  465)
--- [ END: cat /var/log/chkrootkit/log.today ] ---

To create this file containing all output from today's run, do (as root)
# cp -a /var/log/chkrootkit/log.today /var/log/chkrootkit/log.expected
# (note that unedited output is in /var/log/chkrootkit/log.today.raw)
```

Ve výpisu si můžete všimnout poplachů jednak v podobě výše zmíněného otevřeného portu 465 a jednak v podobě několika „podezřelých" souborů. V mém případě jsem dospěl k závěru, že se ve všech případech jedná o poplachy falešné (vám ovšem důrazně doporučuji každý poplach prozkoumat a učinit vlastní závěr). Abyste svůj aktuální stav uložili, je třeba jej zkopírovat do souboru /var/log/chkrootkit/log.expected, tak, jak naznačuje samotný výpis, tedy:

```
cp -a /var/log/chkrootkit/log.today /var/log/chkrootkit/log.expected
```

Od této chvíle bude chkrootkit při denní kontrole brát v úvahu pouze odchylky od tohoto uloženého stavu.

### rkhunter

Druhým detektorem, který představím, je rkhunter. Tento nástroj se zdá být o něco komplexnějším, obsahuje aktualizovatelnou databázi známých hrozeb a vytváří si vlastní databázi kontrolních součtů kritických souborů, se kterou porovnává aktuální stav. Instalaci lze vyřešit podobně jako v případě nástroje chkrootkit:

```
aptitude install rkhunter
```

Prověření systému provedete následujícím příkazem:

```
rkhunter -c
```

Objevíte-li během prověřování nějaké varování, více informací se dozvíte z logu, který se standardně ukládá do /var/log/rkhunter.log. Opět připomínám vhodnost prověření každého poplachu, byť některé z nich mohou být falešné. Stejně jako v případě nástroje chkrootkit, doporučuji si projít informace o falešných varováních v /usr/share/doc/rkhunter/README.Debian.gz. Momentálně existuje falešný poplach na Debianu Squeeze v podobě rootkitu „Xzibit" z důvodu existence řetězce „hdparm" v souboru/etc/init.d/hdparm. Pokud víte, že je to určitě i váš případ, můžete tomuto varování zamezit úpravou konfiguračního souboru /etc/rkhunter.conf, konkrétně přidáním výše zmíněného init skriptu do whitelistu (proměnná RTKT_FILE_WHITELIST).

Nástroj rkhunter obsahuje široké možnosti konfigurace, doporučuji si výše zmíněný konfigurační soubor projít a upravit podle svých představ. Mezi jednu z důležitých voleb patří ALLOW_SSH_ROOT_USER, která by měla být nastavena stejně jako příslušná hodnota v nastavení SSH démona. Rkhunter pak varuje, pokud se hodnota změní. Některé testy jsou v případě Debianu implicitně vypnuty. Naleznete je i spolu s popisem u volby DISABLE_TESTS. Můžete také definovat vlastní hashovací nástroj (výchozí je sha1sum), a to ve volběHASH_FUNC.

Zmínil jsem, že rkhunter obsahuje aktualizovatelnou databázi známých hrozeb. Aktualizaci můžete provést ručně příkazem:

```
rkhunter --update
```

V Debianu se tato databáze aktualizuje automaticky každý týden (viz /etc/cron.weekly/rkhunter). Databázi hashí kritických souborů lze aktualizovat pouze ručně, ideálně pouze jste-li si jisti, že všechny ohlášené změny jsou v pořádku. Aktualizaci můžete provést následujícím příkazem:

```
rkhunter --propupd
```

Stejně jako nástroj chkrootkit má i rkhunter skript pro spouštění cronem v /etc/cron.daily/rkhunter. Jeho nastavení naleznete v souboru /etc/default/rkhunter. Ve výchozím stavu je denní kontrola se zasíláním e-mailů zapnuta.

Na závěr dodám, že rkhunter upozorňuje i na staré (a zranitelné) verze některých démonů a nástrojů. Zde je potřeba mít na paměti, že software v repozitářích distribuce se sice záplatuje, ale verze zůstávají stejné. V tomto případě tedy stará verze nevadí.

## Zotavení po průniku

Obecnou radou, jak si počínat při zjištění průniku, je nepanikařit a nedělat unáhlená rozhodnutí. Pokud již útočník do vašeho systému pronikl, existuje velká pravděpodobnost, že již provedl, co provést chtěl, tj. škoda byla napáchána, tudíž rychlé protiopatření (např. odstavení serveru) jí už nezabrání. Pokud server odstavíte hned nebo až třeba za hodinu, tedy nehraje takovou roli. V podnikovém prostředí často existují předem dané postupy, jak si s takovou situací poradit, koho kontaktovat a jak postupovat. Nebývá na škodu mít takový scénář připraven i v případě menších či osobních serverů.

Kritické je zjistit, kudy útočník do systému pronikl. Pokud to nezjistíte, můžete ho v systému za čas opět najít. Kompromitovanému systému a veškerému softwaru na něm je vhodné nevěřit. Nelze než doporučit kompletní reinstalaci systému, neboť nikdy nebudete mít jistotu, že jste zachytili vše. Před tím je ovšem vhodné udělat zálohu kompromitovaného systému i všech dat k pozdější analýze (či jako důkaz pro případné soudní řízení).

Zotavení napadeného systému je třeba přizpůsobit situaci. Máte-li soukromý server, na kterém veskrze nic důležitého neběží, můžete si dovolit jej odstavit i na delší čas. Jedná-li se ovšem o server, na kterém stojí vaše podnikání nebo přežití vaší firmy, je třeba postupovat opatrně a s rozmyslem.

## Správa linuxového serveru: Sledování změn v souborech pomocí AIDE

V minulém díle jsem probral detektory rootkitů a obecné rady, jak postupovat v případě průniku útočníka do systému. Zajímavými nástroji, které mohou částečně pomoci při detekci, ale hlavně po případném průniku, jsou nástroje sledující změny v souborech. Jedním z těchto nástrojů je AIDE, kterým se bude zabývat tento díl.

### Úvod

Nástroje pro sledování změn v souborech pracují obvykle velmi jednoduše. Vytvoří si databázi hashí jednotlivých souborů (které to jsou, to se obvykle dá nastavit) a s tou pak porovnávají současný stav. Pokud tedy útočník změní některé soubory, změní se i jejich hash, což by mělo vyvolat poplach. Zde je ovšem třeba jednoznačně zdůraznit, že kompromitovanému systému (tj. nástrojům a datům na něm) nelze věřit. Měl-li útočník možnost pozměnit systémové nástroje jako ls, ps, netstat apod., měl i příležitost pozměnit libovolné jiné nástroje a jejich data, včetně těch určených ke sledování změn v souborech. V ideálním případě by databáze měla být pořízena před vypuštěním serveru na síť a bezpečně uložena, přičemž kontroly by měly běžet offline, tj. např. z live CD. V praxi je to obtížně proveditelné, zejména pak vzhledem k tomu, že po aplikování nutných aktualizací dochází ke změnám v souborech, čímž databáze zastarává. Jde to samozřejmě řešit provedením kontroly před aktualizací, provedením aktualizace a následně aktualizací či vygenerováním nové databáze hashí. To je ale samozřejmě nepraktické.

AIDE v Debianu je zkompilováno staticky, což znamená, že samotné binárky nevyužívají systémové knihovny. To sice útočníkovi zkomplikuje život, jelikož změnou příslušných knihoven nedojde k ovlivnění AIDE, ale není to nic, co by se nedalo překonat (rootkit upravující funkčnost jádra, úprava AIDE samotného). Budete-li tedy AIDE používat jiným než výše naznačeným ideálním způsobem, musíte počítat s tím, že chytrý útočník obejde i tento nástroj a vy se o změnách nedozvíte.

Jelikož AIDE používá celou řadu hashovacích algoritmů pro jednotlivé soubory, jeho běh je náročný na procesor i čas. Budete-li jej pouštět ručně na běžícím serveru, zvažte úpravu jeho priority (nástroj nice, popř. ještě ionice).

### Instalace

Instalace AIDE je jednoduchá, stačí využít správce balíčků, v Debianu je možné použít následující příkaz:

```
aptitude install aide
```

Upozorňuji, že pokud byste chtěli dynamicky sestavenou binárku AIDE místo statické, budete muset nainstalovat balíček aide-dynamic. To ale s největší pravděpodobností asi chtít nebudete.

### Konfigurace

Sledujete-li tento seriál pravidelně, popřípadě znáte-li Debian, asi už tušíte, že Debian si konfiguraci AIDE upravuje po svém. V případě AIDE je konfigurace přizpůsobena poměrně značně. Jde to tak daleko, že konfigurační soubor je sestavován skriptem update-aide.conf, který sesbírá všechna nastavení ze souborů uložených v /etc/aide/aide.conf.d a zapíše je do jediného konfiguračního souboru, který pak využívá „obalovač" (wrapper) AIDE, tedy aide.wrapper.

Za základ konfigurace je třeba považovat soubor /etc/default/aide, který obsahuje nastavení pro příslušnou úlohu cronu, jež provádí denní kontroly. Tento soubor je dobře zdokumentovaný, takže si jej projděte a nastavte podle svých potřeb.

Za volby hodné zvláštní pozornosti považuji následující:

- MAILSUBJ - předmět zasílaných e-mailů
- MAILTO - komu posílat e-maily
- QUIETREPORTS - potlačení zaslání e-mailu, pokud nedošlo ke změně (je-li tato proměnná nastavena nayes)
- TRUNCATEDETAILS - zkrácení detailních informací o změnách
- FILTERUPDATES - potlačení informování o změnách způsobených aktualizacemi (předpokládáTRUNCATEDETAILS nastaveno na yes)

AIDE je v Debianu nastaveno tak, že prohledává poměrně velkou část souborového systému, včetně adresáře /home, což může působit řadu zbytečných „poplachů", máte-li aktivní uživatele. Nechcete-li, aby AIDE prohledávalo /home, můžete využít ukázkového skriptu v /usr/share/doc/aide-common/examples a zařadit jej do konfigurace, takto:

```
cp /usr/share/doc/aide-common/examples/31_example_exclude-homes /etc/aide/aide.conf.d/
chmod a+x /usr/share/doc/aide-common/examples/31_example_exclude-homes
```

AIDE také prohledává cache správce balíčků Apt, což rovněž nemusí být zrovna to, co chcete. Nechcete-li, aby AIDE brala v úvahu uložené balíčky ve /var/cache/apt/archives, upravte soubor/etc/aide/aide.settings.d/31_aide_apt_settings, ve kterém nastavte proměnnou IGNORE_ARCHIVES na"yes":

### Inicializace databáze

Až budete hotovi s nastavováním (ne dříve!), můžete inicializovat databázi AIDE:

```
aideinit
```

Po případných úpravách konfigurace je vhodné znovu inicializovat databázi AIDE, takto:

```
update-aide.conf && aideinit -y -f
```

### Manuální spuštění AIDE

Chcete-li zkontrolovat integritu databáze, nejjednodušší je pustit samotnou úlohu pro cron:

```
/etc/cron.daily/aide
```

Chcete-li použít samotný nástroj aide, použijte příslušný „obalovač", který zajistí, že se použijí správně sestavené konfigurační soubory. Tento „obalovač" je k dispozici jako aide.wrapper.

### Úprava konfigurace

Nepochybně zjistíte, že AIDE v Debianu prohledává kdeco a budete tedy chtít některé soubory či adresáře z prohledávání vyřadit. Rovnou předešlu, že úprava nastavení není úplně triviální, jelikož Debian nabízí opravdu značné množství konfiguračních souborů i skriptů v /etc/aide/aide.conf.d, které se pak spouští, spojují a kompletují do jediného velkého konfiguračního souboru.

Zde doporučuji prostudovat manuál k AIDE (a ještě manuál k aide.conf: man aide.conf), jelikož obsahuje příslušnou syntax a potřebné informace. Zásadní informace je, že jednotlivá pravidla jsou de facto regulární výrazy, čemuž byste měli přizpůsobit psaní pravidel, zejména si dejte pozor na tečky (tečka odpovídá jednomu libovolnému znaku), které raději escapujte (umístěním zpětného lomítka před tečku, takto: \.). Regulárními výrazy se zde zabývat nebudu, avšak pod článkem jsem vám připravil odkazy na česky psané materiály, ze kterých se příslušnou syntaxi můžete snadno a rychle naučit.

Co se týká pořadí pravidel (opět odkazuji na manuál), je vhodné obecnější pravidla umisťovat níže a specifičtější pravidla výše. Asi nejčastější úpravou konfigurace bude specifikace souborů, které nechcete, aby se do databáze zařadily. V takovém případě si vytvořte vhodný soubor v /etc/aide/aide.conf.d, ale jelikož záleží na pořadí, zařaďte jej do struktury pomocí úvodního čísla, tzn. např. 50_aide_moje-pravidla. Do tohoto souboru pak přidejte příslušná pravidla - pro ignorování určitého souboru zapište příslušný regulární výraz z vykřičníkem na začátku, např. takto:

```
!/var/cache/munin/www
```

AIDE pak bude takový soubor či adresář ignorovat. AIDE v zásadě podporuje tři typy pravidel, a sice regulární výrazy (začínají lomítkem), přesné specifikace konkrétního souboru či adresáře ("equal matching", začínají znakem =) a vylučovací regulární výrazy (začínají vykřičníkem). U pravidel, která zařazují soubory či adresáře do databáze, je třeba specifikovat parametr, který odpovídá operacím, jež se mají s daným souborem provést. Zde opět odkazuji na manuál, kde jsou všechny popsány. Můžete si ušetřit práci tím, že si sadu operací pojmenujete, a pak se na ni budete odkazovat, takto:

```
MojeAkce = p+i+MojeJineAkce
```

Jak si můžete všimnout, ve specifikaci vlastních operací se můžete odkazovat na jiné vlastní operace. Konfigurace Debianu používá řadu vlastních pojmenovaných operací, které naleznete v/etc/aide/aide.conf. Jedno z pravidel je např. Full, které kdybyste chtěli aplikovat na konkrétní soubor či adresář, zapíšete:

```
/home/secret$ Full
```

Pokud by v tomto případě byl secret adresář, zaznamenaly by se jen jeho parametry, ale soubory v něm obsažené by se ignorovaly. Pokud byste chtěli přidat i jeho obsah, stačí vynechat znak konce řádky v regulárním výrazu (tedy znak dolaru), takto:

```
/home/secret Full
```

Tolik k běžným úpravám nastavení AIDE. Nezapomeňte, že po úpravě konfigurace musíte znovu inicializovat databázi, viz sekce *Inicializace databáze* výše.

## Správa linuxového serveru: Úvod do kompilace softwaru

Repozitáře linuxových distribucí obsahují ohromné množství softwaru. Co když ale potřebujete komponentu, která v dostupných repozitářích není, popřípadě je, ale v jiné verzi, než jakou požadujete? V takovém případě vám nezbude než se pustit do kompilace ze zdrojových kódů. V tomto díle se dozvíte o správě softwaru v linuxových distribucích, kompilaci obecně a alternativách ke kompilaci.

### Balíček, repozitář, závislost, správce balíčků

Toto jsou pojmy, které – předpokládám – dobře znáte, tudíž je prolétnu jen letem světem. Balíček je obecně komponenta. Může to být aplikace, ale také dokumentace, datové soubory, sada ikon, fontů či tapet, knihovna atd. Jelikož balíčky mají atomický charakter (obsahují „jedno kolečko v soukolí"), existují mezi nimi závislostní vazby, tj. jeden balíček pro svou funkci může vyžadovat další balíčky, a ty mohou vyžadovat další balíčky atd.

Balíčky jsou uloženy v repozitářích. To jsou jednoduše skladiště balíčků. Mohou mít podobu adresáře, síťového serveru, instalačního média atd. Nejčastěji se používají síťová úložiště.

Software je v linuxových distribucích spravován centrálně, prostřednictvím správce balíčků, který bere v úvahu závislosti, ví, co je dostupné v repozitářích a de facto provádí téměř vše za správce. Ten mu jen řekne, co chce instalovat, odinstalovat či aktualizovat, a správce balíčků to provede.

### Kompilace

Software pro Linux, alespoň ten svobodný a open-source, je obvykle k dispozici v podobě zdrojového kódu (člověkem čitelné podoby programu). Ten je v případě řady programovacích jazyků nutné převést do podoby, kterou je schopen zpracovat procesor, tedy do strojového kódu. Ten pak můžete spustit a využívat.

V případě většiny linuxových distribucí tento proces již někdo provedl a do repozitářů umístil příslušný balíček se strojovým kódem (někdy také „binárkou"), který stačí nainstalovat a je připraven k použití.

### Distribuce a software

Centralizovaná správa softwaru, tedy správa softwaru pomocí správce balíčků, je považována za jednu ze silných stránek GNU/Linuxu. Má ovšem z pohledu správce serverů dvě nevýhody. V první řadě, ne všechen software dostupný pro GNU/Linux je v distribučních repozitářích a ne vždy je to proto, že by daná věc nebyla zajímavá či užitečná. Abych nezůstával příliš v teoretických rovinách, uvedu malý příklad. Debian (a řada dalších distribucí) neobsahuje jádra patchovaná pomocí grsecurity. Zřejmě proto, že se dosud nenašel dobrovolník, který by jádra připravoval a udržoval. Řada distribucí, včetně Debianu, stojí na ryze komunitním vývoji. O konkrétní balíčky se starají konkrétní lidé (říká se jim maintainer), kteří jsou zodpovědní za jejich tvorbu i údržbu. Někdy se stane, že nějaký balíček z repozitářů vypadne, nebo nebude začleněn, protože to představuje hodně práce a nikomu se do toho nechce. U podnikových distribucí, kde není problém na „nechtěnou" práci někoho najmout, je zase typické, že jejich repozitáře obsahují méně softwaru než jejich ryze komunitní kolegové.

Druhým bodem, který lze vnímat v závislosti na úhlu pohledu jako výhodu i jako nevýhodu, je zmrazení verzí. Když distribuce s klasickým modelem vývoje vydá nové vydání, v repozitářích zpravidla zůstanou jednotlivé komponenty ve stejných verzích po celou dobu podpory daného vydání. Je-li v nějakém balíčku nalezena chyba, která byla opravena v nové verzi dané komponenty, provede se tzv. backport příslušné změny, tj. změna se aplikuje na starší verzi komponenty, která je dostupná v repozitářích. Do repozitářů se tedy nedostávají novější verze komponent, a ty samozřejmě časem zastarávají.

Tento systém má svůj dobrý důvod – nové verze komponent obsahují změny, které nemusí být vždy zpětně kompatibilní. U serverového softwaru se může měnit syntax konfiguračních souborů i chování samotných démonů, čímž může vzniknout jak potenciální bezpečnostní zranitelnost, tak nepříjemný stav, kdy se po aktualizaci démon odmítne znovu spustit. Nehledě na to, že nové verze obsahují zpravidla kromě nových zajímavých vlastností i nové chyby.

Dlužno dodat, že tento problém se netýká distribucí, které používají tzv. rolling release model vývoje (Arch Linux, Gentoo, Source Mage GNU/Linux apod.), kde se nové verze (po otestování) rovnou zařazují do hlavních repozitářů, a distribuce jsou tak vlastně ve stádiu neustálého vývoje. Tyto distribuce obecně nebývají doporučovány pro produkční servery, i když jsou na některých používány. Jistě, záleží v první řadě na tom, co a jak má na serveru běžet a která distribuce je pro to nejvhodnější (nehledě na to, že velkou roli hraje i to, kterou distribuci správce dobře ovládá).

### Úskalí kompilace

Balíčkovací systém šetří (nejen) správcům práci. Umožňuje snadnou instalaci, odinstalaci i aktualizaci softwaru. Aktualizace se týká veškerého softwaru dostupného z repozitářů (pokud distribuce nestanoví jinak). Ve chvíli, kdy se pustíte do kompilace, o tyto výhody přijdete. Pokud si nesestavíte vlastní balíček obsahující vámi zkompilovaný software, nebude moci brát balíčkovací systém v úvahu soubory, které tomuto softwaru v souborovému systému patří - odinstalování tedy může být problematické. Ale i když si vlastní balíček postavíte a nainstalujete jej přes správce balíčků, zůstane na vás stále to nejdůležitější - jeho aktualizace.

#### *Závislosti*

Samotná kompilace není jednoduchá. Nejenom, že potřebujete kompilátory a příslušné nástroje, ale musíte také vyřešit závislostní vazby. Je možné, že váš software vyžaduje knihovny nebo komponenty, které v systému nemáte. Ty je pak třeba buď doinstalovat z repozitářů, nebo také zkompilovat.

#### *Změny v knihovnách*

Aktualizovat balíček byste měli nejenom, když vyjde jeho nová verze, ale také, pokud se zásadně změní verze závislých knihoven. Tohle je ale naštěstí spíše problém rolling release distribucí, jelikož v klasických zůstanou knihovny zpravidla ve stejných verzích. Může ale nastat problém při povýšení distribuce. V rolling release distribucích je třeba překompilovávat častěji, neboť tam se verze knihoven čas od času mění.

### Alternativy

Ideální je, pokud jste schopni pracovat se softwarem, který je v distribučních repozitářích, byť někdy musíte překousnout, že nová verze, už třeba půl roku dostupná, umí něco nového, co by vám usnadnilo práci. Ale ideální situace nenastane vždy – proto se někdy kompilaci nevyhnete. Ovšem ještě před tím, než se do kompilace pustíte, by bylo vhodné zvážit možná alternativní řešení.

#### *Debian backports*

Jelikož se tento seriál zaměřuje na distribuci Debian, nelze nezmínit nyní již oficiální projekt Debian backports. Tento projekt tvoří repozitář, který lze zkombinovat se standardními repozitáři Debianu, a získat tak možnost instalace vybraného novějšího softwaru, který byl zkompilován pro stabilní vydání. Kupříkladu, kdybyste nebyli spokojeni s Postgresem 8.4 v Debianu Squeeze a chtěli Postgresql 9.0, který nedávno vyšel, nemuseli byste sáhnout ke kompilaci – stačilo by vám použít backporty, ze kterých byste verzi 9.0 nainstalovali.

#### *Neoficiální repozitáře*

Řada projektů má k dispozici neoficiální repozitáře (neoficiální z pohledu Debianu), které můžete použít k získání aktuální verze ve stabilním vydání. Kupříkladu, zde v seriálu zmíněný Jabber/XMPP server Prosody má k dispozici neoficiální repozitář, kde najdete vždy aktuální verzi.

### Strategie

Jelikož kompilace není jednoduchá a má svá úskalí, bývá dobré se jí vyhnout, pokud můžete. Zde mohou pomoci výše zmíněné alternativy. Pokud se pro kompilaci rozhodnete, musíte zvážit jednak dopad na „pohodlnost" správy, respektive na nové povinnosti a možná úskalí, ale také na možné bezpečnostní dopady (síťová služba, SUID root popřípadě pod rootem běžící atd.). Řada projektů má k dispozici e-mailové konference s bezpečnostními hlášeními nebo hlášeními o nových verzích. Doporučuji se do nich přihlásit a pravidelně číst poštu, abyste mohli reagovat na případný bezpečnostní problém nebo věděli o dostupnosti nové verze.

Tím bych tento díl ukončil. Příště vám kompilaci představím po praktické stránce.

## Správa linuxového serveru: Kompilace softwaru

V minulém díle byla probrána teorie kolem kompilace softwaru včetně alternativ, pomocí kterých se můžete kompilaci vyhnout. Tento díl se bude věnovat praktické stránce kompilace, i když zatím převážně jen teoreticky, stejně jako jejím úskalím.

### Úvod

Poselstvím minulého dílu měl být pohled na kompilaci spíše jako na nouzové řešení. Ideální je používat správce balíčků a software instalovat a spravovat prostřednictvím něj. Ruční kompilace je nejen relativně složitější, ale také vyžaduje, abyste se o takto instalovaný software starali, tedy primárně sledovali bezpečnostní problémy, hodnotili jejich vážnost a řešili je, druhotně pak abyste také měli přehled o dostupnosti nových verzí a změn fungování i nastavování daného softwaru. To je velmi důležité, jelikož nové verze mohou měnit syntax či parametry v konfiguračních souborech, což by při slepém přechodu na novou verzi mohlo zapříčinit nefunkčnost dané komponenty.

Kompilace však přináší i jisté výhody, které pak dotahují do konce zdrojově založené (source-based) distribuce jako Gentoo, SourceMage GNU/Linux atd., které pak kompilaci v mnohém usnadňují. Co se týče výhod kompilace, v první řadě je to jistě možnost mít konkrétní verzi softwaru bez ohledu na to, která verze je k dispozici v aktuálním vydání distribuce. Vedle toho je to ale také možnost používat upravenou verzi, aplikovat specifické patche nebo naopak možnost použít čistou (tzv. vanilla) verzi, která nebyla distribučně upravena – řada distribucí totiž pro různé komponenty využívá vlastní patche, které mohou měnit jejich funkčnost, což někdy přináší nové vlastnosti, ale někdy to může být zdrojem chyb či problémů (v souvislosti s tím nemohu nezmínit nechvalně známý zásah Debianu do openssl knihovny, který díky predikovatelnosti generátoru náhodných čísel oslabil veškeré klíče či certifikáty generované pomocí této knihovny).

Leckterý software lze sestavit s podporou nějaké komponenty nebo bez ní. Kupříkladu, oblíbený textový editor Vim může fungovat jak v terminálu, tak samostatně v grafickém prostředí. Podpora grafického režimu je otázkou kompilační volby. Na serveru je obvykle zbytečné mít konzolové aplikace zkompilované s podporou Xorg, což u binárních distribucí znamená mj. nutnost dotáhnout spolu s instalovanými aplikacemi ještě Xkové knihovny. Kompilací můžete sami určit, jak se má vámi instalovaný software sestavit. Výše uvedená situace je opravdu jen příklad pro ilustraci, jelikož v Debianu můžete tuto situaci snadno vyřešit nainstalováním verze Vimu zkompilované bez Xorg (balíček vim-nox).

Další výhodou kompilace je možnost použít specifické volby kompilátoru, které mohou kód optimalizovat (zrychlit, zmenšit apod.) či jej lépe zabezpečit.

Máte-li potřebu měnit konfigurační volby a kompilovat větší množství softwaru podle svých potřeb, doporučuji zvážit použití některé ze source-based distribucí.

### Kompilační nástroje a zdrojový kód

Abyste mohli kompilovat, potřebujete příslušné nástroje, zejména pak kompilátor. Pro software napsaný v C nebo C++ postačí GCC, dále pak nástroj make. V Debianu je možné tyto nástroje nainstalovat prostřednictvím metabalíčku build-essential:

```
aptitude install build-essential
```

Ne vždy ale budete kompilovat zrovna aplikace v C. To poznáte podle dokumentace k danému softwaru, která by vám měla říci, co potřebujete k úspěšnému sestavení.

Kromě kompilačních nástrojů potřebujete samozřejmě zdrojový kód daného softwaru. Obvykle jeho získání nepředstavuje problém, pomocí vyhledávače lokalizujete oficiální web projektu, stáhnete příslušný balíček, obvykle tar.gz či tar.bz2, přičemž před jeho použitím ověříte kontrolní součty nebo digitální podpisy. Na serveru se vám určitě hodí nástroj wget, popřípadě nějaký textový prohlížeč (např. lynx), abyste mohli příslušný software stahovat přímo na serveru a nemuseli ho tam kopírovat ze své stanice (což byste nejspíše provedli pomocí nástroje scp či FUSE modulu sshfs).

#### *SCM: Systémy pro správu verzí*

Chcete-li opravdu „bleeding-edge" kód (na serveru lze jeho použití důrazně nedoporučit), přijdete nejspíše do styku s nástroji pro správu verzí (SCM, Source Control Management, tedy např. Git, Mercurial, Subversion apod.). To jsou nástroje, které umožňují v zásadě dvě věci – v první řadě udržují přehled změn ve zdrojovém kódu, ve druhé řadě pak umožňují nebo usnadňují spolupráci více vývojářů na jednom projektu. Z pohledu správce obvykle představují nutnost nainstalovat příslušný nástroj nebo jeho klienta. Zde předešlu, že mnoho webových služeb (GitHub apod.) umožňuje stáhnout balíček s kódem odpovídajícím dané revizi, čímž si tuto práci ušetříte.

### Závislosti a instrukce pro sestavení

Řada softwaru má k dispozici instrukce pro sestavení buď v dokumentaci na webu projektu, nebo v souboru `INSTALL`, který je přibalován ke zdrojovému kódu. Měli byste zde najít veškeré podstatné závislosti.

Závislosti kompilovaných programů lze rozdělit do dvou kategorií – závislosti kompilační, které jsou nutné pro kompilaci, ale už ne pro samotný běh programu, a závislosti běhové, které jsou třeba pro samotnou funkčnost zkompilovaného softwaru. Kompilační závislostí je samozřejmě kompilátor a s ním spojené nástroje, ale občas to může být i něco jiného. Více vám napoví dokumentace k danému softwaru. Podstatné je, že kompilační závislosti můžete po kompilaci bez obav ze systému odstranit, ty běhové samozřejmě nikoliv.

Jelikož v případě kompilace za vás správce balíčků závislosti řešit nemůže, musíte se s nimi vypořádat ručně. Řešit závislosti je však vhodné s pomocí správce balíčků: máte-li příslušnou závislost v požadované verzi v repozitářích, můžete ji nainstalovat a nemusíte ji kompilovat. K vyhledávání můžete použít příslušné funkce správce balíčků:

```
aptitude search závislost
apt-cache search závislost
```

Pokud danou závislost v repozitářích nemáte, nebo ji máte ve staré verzi, nezbude vám než ji zkompilovat také. Vzhledem k tomu, že závislostní vazby mohou být košaté, musíte zvážit, zdali se vám kompilace, ale také následná údržba, opravdu vyplatí. Máte-li stabilní binární distribuci, která je už nějaký čas venku, může se klidně stát, že aktuální verze vámi zvoleného softwaru bude vyžadovat kompilaci řady dalších komponent, jelikož v repozitářích je sice máte, ale příliš zastaralé. Udržovat takto sestavený software ručně je dřina.

#### *Hlavičkové soubory*

V případě knihoven a komponent, které tvoří závislosti vámi zvoleného softwaru, potřebujete nejenom binárky příslušných knihoven, ale také tzv. hlavičkové soubory. Hlavičkové soubory představují kompilační závislost, k běhu softwaru po kompilaci je tedy nepotřebujete. V případě některých distribucí jsou hlavičkové soubory součástí balíčků s danými komponentami (namátkou Arch Linux). V Debianu jsou umístěny v samostatných balíčcích s přídomkem -dev. Pokud byste tedy měli knihovnu libssl v závislostech vámi zvoleného softwaru, nainstalovali byste nejenom balíček libssl, ale také libssl-dev.

### Svatá trojice

Tento díl zakončím základní informací o tzv. svaté trojici, tedy sérii třech příkazů, které se nejčastěji zmiňují v souvislosti s kompilací. V příštím díle na tuto informaci navážu a ukážu vám vše na příkladu. „Svatá trojice" vypadá takto:

```
./configure
make
make install
```

První příkaz spouští skript configure v aktuálním adresáři. Předpokládá se, že se nacházíte v adresáři s rozbalenými zdrojovými kódy. Tento skript je generován obvykle vývojářem, pomocí nástroje autoconf (více viz GNU build system), a distribuován spolu se zdrojovými kódy. V některých případech (často při kompilaci softwaru získaného pomocí SCM) se můžete setkat s tím, že tento skript chybí a vy si jej musíte vygenerovat sami. Skript configure analyzuje váš systém a vygeneruje soubor Makefile, který pak řídí proces kompilace. Pokud vám chybí nějaká závislost, nebo váš systém neodpovídá požadavkům daného softwaru, měl by tento skript skončit a vyprodukovat chybovou hlášku. Ta by vám měla osvětlit, co vám chybí nebo co musíte udělat.

Jakmile máte k dispozici Makefile, můžete spustit kompilaci pomocí nástroje make. Pokud půjde vše dobře, kompilace skončí úspěchem. Zbývá už jen software nainstalovat. Jeho instalaci je možné provést příkazem make install, avšak tento postup nedoporučuji – software instalovaný mimo repozitáře je vhodné izolovat od zbytku systému, abyste měli přehled o tom, kde se nachází jeho soubory a nezanesli si do systému nepořádek. Popřípadě si můžete (např. pomocí nástroje checkinstall) vytvořit balíček, který pak nainstalujete pomocí správce balíčků (v Debianu nástrojem dpkg). O tom ale až v příštím díle.

## Správa linuxového serveru: Kompilace softwaru prakticky

Ačkoliv je ideální používat správce balíčků a repozitáře, které obsahují drtivou většinu toho, co budete potřebovat, někdy není jiná možnost a je třeba sáhnout ke kompilaci nějakého softwaru ze zdrojových kódů. Poslední dva díly se věnovaly kompilaci převážně z teoretického hlediska, tento díl vše ukáže na praktickém příkladě.

### Úvod

Pokud jste si nepřečetli poslední dva díly (první díl, druhý díl), doporučuji tak učinit, neboť v nich zazněly velmi podstatné informace, počínaje popisem správy softwaru v linuxových distribucích, přes různé alternativy ke kompilaci či důvody, proč je vhodné se jí snažit vyhnout, až po samotný postup a technické pozadí kompilace, které bylo představeno v minulém díle.

### Stabilní, vývojová a SCM verze

Ještě před samotnou kompilací zmíním jednu věc, kterou jsem dostatečně neosvětlil v minulých dvou dílech, a sice, jaké verze softwaru jsou obvykle k dispozici a jaký je mezi nimi rozdíl. Většina FOSS projektů má něco jako stabilní vývojovou větev. Stabilní větev by měla obsahovat dostatečně otestovaný, časem ověřený kód s minimem chyb. Na server je to určitě dobrá volba.

Často je ovšem k dispozici i testovací či vývojová verze. Ta obsahuje obvykle novější vlastnosti, které zatím ještě do stabilní verze nepronikly, neboť nebyly dostatečně otestovány. Vývojové verze obsahují zpravidla řádově více chyb než jejich stabilní protějšky, přičemž některé vlastnosti ve vývojových verzích nemusí být ještě zcela funkční (nebyly zcela implementovány), popřípadě se mohou časem i zásadně měnit – na to je třeba si dávat obzvlášť pozor, zejména při aktualizacích. Vývojovým verzím je vhodné se vyhnout, pokud to jde.

V souvislosti s vývojovými verzemi je vhodné zmínit i často používané názvosloví určující aktuální stádium vývoje: pre-alpha, alpha, beta a release candidate. Verze označená jako pre-alpha je obvykle software v raném stádiu vývoje, mnohé vlastnosti ještě nejsou vůbec implementované, mnohé se mění, často i zásadně. Taková verze by na serveru neměla co dělat, jelikož může obsahovat kritické chyby vedoucí ke ztrátě dat. Stádium alpha je na tom jen o málo lépe než pre-alpha. Mnohé vlastnosti by již měly být implementovány, i když stále je pravděpodobný výskyt kritických chyb. Ledacos také nemusí fungovat správně nebo se může časem výrazně změnit. I toto stádium bych na server důrazně nedoporučil. Stádium beta by mělo označovat dokončení všech připravovaných vlastností (ty by se již neměly měnit) a počátek testování ze strany uživatelů. Chyby jsou, přirozeně, v tomto stádiu stále očekávané. Následuje release candidate, tedy kandidát pro vydání. To už je více méně hotový produkt, ale stále se čeká na nalezení závažných chyb. Pokud nejsou nalezeny závažnější chyby během určité doby, obvykle dojde k vydání.

Poslední „verze" softwaru, se kterou můžete přijít do styku, je snapshot ze SCM nástroje (Subversion, Git, Mercurial, apod.). Je to de facto snímek zdrojového kódu platný k určitému datu – nejčastěji se bere ten nejnovější. Jelikož pokročilé nástroje pro správu verzí softwaru (SCM) umí držet současně více větví, záleží určitě, kterou větev (branch) jste si zvolili jako zdroj (záleží ovšem také na tom, jaká pravidla vývoje jsou v rámci daného projektu v platnosti). Stabilní větev by měla mít přibližně charakter stabilního vydání, vývojová větev bude v některém z výše uvedených stavů, a její nasazení na server lze proto důrazně nedoporučit. Často se ovšem s takovou verzí setkáte, jelikož v ní bude patrně opravená nějaká nepříjemná chyba (oproti poslední vývojové verzi stažitelné z webu projektu, pokud taková je).

Výše uvedená pravidla samozřejmě podléhají tomu, jak na vývoj nahlíží vývojáři projektu, který chcete nasadit. Stabilní verze jednoho projektu může stabilitou klidně odpovídat vývojové verzi jiného. Záleží na typu projektu a na vývojářích. Obecně je vhodné se nestabilním verzím vyhnout, pokud nemáte opravdu dobrý důvod je nasadit.

### Kompilace zvoleného softwaru

Software, který bude sloužit jako pokusný králík pro kompilaci, je webový server Apache, kterému se tento seriál věnoval v několika dílech. Je to pouze příklad, na kterém bude demonstrována kompilace. V žádném případě se nejedná o doporučený způsob kompilace Apache. Kromě toho, Apache určitě naleznete v repozitářích vámi používané distribuce. Budete-li chtít kompilovat právě Apache, určitě se primárně řiďte oficiální dokumentací, na kterou článek pro úplnost odkazuje.

### Nahlédnutí do dokumentace

Před tím, než budete software kompilovat, doporučuji podívat se do dokumentace, ať už dokumentace na webu projektu nebo dokumentace přibalené v příslušném zdrojovém balíčku. Obvykle zde naleznete návody či případné důležité informace týkající se kompilace ze zdrojových kódů. Uvnitř balíčku hledejte soubory README a INSTALL. V případě webového serveru Apache je dokumentace dostupná online.

### Získání a příprava zdrojového kódu

Vyberte si vhodnou verzi (dle rad výše) a stáhněte si příslušný balíček, obvykle je to tar.gz, i když tar.bz2bývá menší, tedy úspornější na přenášená data. Na serveru můžete pro samotné stažení použít nástrojwget, což se určitě hodí, máte-li na stanici, ze které se vzdáleně připojujete k serveru, pomalý upload (typické pro ADSL a mobilní připojení). Můžete si také pomoci textovým prohlížečem přímo na serveru:

```
lynx http://httpd.apache.org/download.cgi
```

Jelikož je při instalaci na server vhodné dbát zvýšené opatrnosti, je třeba si po získání balíčku ověřit kontrolní součty či digitální podpisy. Opatřete si příslušné PGP klíče a ověřte podpis náležející příslušnému souboru. Zde je samozřejmě otázka, jak si opatřit PGP klíče vývojářů z opravdu důvěryhodného zdroje. Pokud se vám nechce hledat nejbližšího vývojáře, spojit se s ním, domluvit osobní setkání, ověřit jeho občanku a zaznamenat si otisk (fingerprint) jeho klíče, můžete zkusit méně bezpečnou metodu popsanouna stránkách projektu, a sice stáhnout soubor s PGP klíči vývojářů (doporučuji alespoň přes HTTPS a přímo ze serveru apache.org), importovat tyto klíče a ověřit podpis, takto:

```
gpg --import KEYS
gpg --verify httpd-2.2.20.tar.bz2.asc
```

Po ověření balíček se zdrojovým kódem rozbalte:

```
tar xvf httpd-2.2.20.tar.bz2
```

### Závislosti

Závislosti, ať již kompilační či běhové (vysvětlení v minulém díle) si budete muset zajistit sami (ideálně instalací příslušných balíčků, v horším případě je budete muset zkompilovat jako první). V rámci Debianu je třeba dodat, že hlavičkové soubory potřebné pro kompilaci jsou v extra balíčcích. Pokud tedy máte jako závislost nějakou komponentu, obvykle nestačí instalovat pouze balíček s binárkou komponenty (např.libknihovna), ale musíte ještě přidat balíček s příslušnými hlavičkovými soubory pro kompilaci, tedy např.libknihovna-dev.

Nejideálnějším postupem je zjistit závislosti z dokumentace, nejméně účinným způsobem je pak spouštět skript ./configure opakovaně a orientovat se dle chybových hlášek. V Debianu však existuje ještě jeden elegantní způsob, jak závislosti řešit. Máte-li vámi kompilovaný software v repozitářích (byť v jiné verzi), můžete si ušetřit práci pomocí správce balíčků – Debian totiž umožňuje kompilaci z tzv. zdrojových balíčků, a v souvislosti s tím umožňuje jednoduchým příkazem nainstalovat všechny závislosti potřebné pro kompilaci daného balíčku, takto:

```
apt-get build-dep apache2
```

Chcete-li pouze rekompilovat balíček dostupný z repozitářů (nechcete jinou verzi), popřípadě chcete-li si sestavit vlastní balíček a hledáte vhodný vzor, můžete použít apt-src, postup je naznačen v odkazech pod článkem.

### configure

Následuje „svatá trojice", tedy ./configure, make a make install. Fáze konfigurace je ta nejnáročnější na činnost správce, jelikož se obvykle nespokojíte s pouhým spuštěním skriptu ./configure, ale budete hledat ty správné volby, které daný program zkompilují s těmi správnými vlastnostmi. Mnohé vlastnosti lze totiž zahrnout nebo naopak vyřadit vhodnými parametry. Méně vlastností znamená obvykle větší bezpečnost, naopak někdy můžete mít zájem zkompilovat daný software s parametrem, který vámi zvolená distribuce při sestavování binárního balíčku ignorovala. Zde opět odkážu na dokumentaci vámi zvoleného projektu. Dokumentace Apache je nejen v tomto ohledu vskutku vynikající, viz podrobný článek o konfiguračních volbách. Kupříkladu, pokud nechcete v Apachi používat CGI, použijete volbu --disable-cgi:

```
./configure --disable-cgi
```

Nejzásadnější volbou pro ./configure je patrně --prefix, která určuje, kam se daný software nainstaluje ve fázi make install. Specifikací konkrétního adresáře můžete klidně udržovat více verzí daného softwaru najednou:

```
./configure --prefix=/opt/apache-2.2.20
```

Pokud nespecifikujete prefix, použije se výchozí volba, která závisí na tom, co zvolil příslušný vývojář. Nejčastěji je cílem /usr/local. Já však, jak naznačuje příklad výše, preferuji /opt. To ale není důležité. Důležité je, abyste měli o souborech, které se vytvoří při instalaci, přehled, a mohli pak software snadno odinstalovat nebo spolehlivě nahradit novou verzí.

Unixový software si obvykle v adresáři specifikovaném jako prefix vytvoří vlastní strukturu s adresáři bin,lib, atd., a tam umístí příslušné soubory. V mém příkladu výše byste pak ke zkompilovanému Apachi přistupovali přes /opt/apache-2.2.20/bin/apache2ctl.

## make

Tento příkaz provede samotnou kompilaci. Máte-li vícejádrový či víceprocesorový server (nebo obojí), doporučuji přidat volbu `-j` s parametrem v podobě čísla odpovídajícímu počtu procesorů zvýšeného o jedničku. Máte-li dvouprocesorový server se čtyřmi jádry, tj. s osmi logickými procesory, zvolili byste devítku (8 jader + 1):

```
make -j9
```

V době vícejádrových a vícevláknových procesorů by byla škoda kompilovat na jednom jádře, tedy velmi, velmi pomalu v porovnání s možnostmi stroje. Na serveru, který obsluhuje klienty, je však vhodné ještě upravit prioritu kompilace. V unixových systémech se priorita nastavuje jako „niceness" (ohleduplnost), tzn. vyšší číslo znamená nižší prioritu (větší ohleduplnost k okolí). V Linuxu má priorita meze od -20 (nejvyšší priorita) do 19 (nejnižší priorita). Pokud na kompilaci nespěcháte, devatenáctka je asi nejvhodnější:

```
nice -n 19 make -j9
```

Máte-li naopak pomalý počítač, který kompiluje příliš dlouho, a na dostatečně rychlé LAN máte k dispozici silný stroj, obraťte svou pozornost k nástroji distcc, kterým můžete propojit více počítačů a vytvořit si malý kompilační „cluster".

## Instalace

Pokud kompilace proběhne bez chyb, zbývá software nainstalovat. Dosud bylo možné všechny kroky (s výjimkou používání správce balíčků) vykonávat bez práv superuživatele. K instalaci je ovšem jeho oprávnění potřeba. Výjimkou může být situace, kdy jste použili prefix k tomu, abyste software mohli nainstalovat do adresáře, který patří aktuálnímu uživateli. V takovém případě bez obav instalujte.

Pokud software nainstalujete přímo, pomocí make install, správce balíčků o těchto souborech nebude nic vědět a nebude je moci odstranit nebo vám říci, kde jsou. Můžete si samozřejmě vytvořit balíček, který nainstalujete pomocí dpkg, ale to obvykle představuje hodně práce. Existuje však i jiná možnost - nástrojcheckinstall, který se používá místo make install. Tento nástroj vám balíček na počkání balíček vytvoří:

```
checkinstall
```

Checkinstall je interaktivní program, stačí zodpovědět několik otázek, případně upravit nastavení balíčku. Poté vytvoří Checkinstall balíček obsahující váš program. Tento balíček můžete nainstalovat pomocí nástroje dpkg:

```
dpkg -i apache-custom_2.2.20-1_i386.deb
```

Checkinstall ovšem nevyplňuje závislostní vazby balíčku. Je to tedy jen polovičaté řešení – správce balíčků sice bude o vašem softwaru vědět a bude ho moci odstranit, ale nebude moci v souvislosti s ním řešit závislostní vazby. Ty si budete muset ohlídat sami.

## Správa linuxového serveru: Kompilace softwaru prakticky

Ačkoliv je ideální používat správce balíčků a repozitáře, které obsahují drtivou většinu toho, co budete potřebovat, někdy není jiná možnost a je třeba sáhnout ke kompilaci nějakého softwaru ze zdrojových kódů. Poslední dva díly se věnovaly kompilaci převážně z teoretického hlediska, tento díl vše ukáže na praktickém příkladě.

### Úvod

Pokud jste si nepřečetli poslední dva díly (první díl, druhý díl), doporučuji tak učinit, neboť v nich zazněly velmi podstatné informace, počínaje popisem správy softwaru v linuxových distribucích, přes různé alternativy ke kompilaci či důvody, proč je vhodné se jí snažit vyhnout, až po samotný postup a technické pozadí kompilace, které bylo představeno v minulém díle.

### Stabilní, vývojová a SCM verze

Ještě před samotnou kompilací zmíním jednu věc, kterou jsem dostatečně neosvětlil v minulých dvou dílech, a sice, jaké verze softwaru jsou obvykle k dispozici a jaký je mezi nimi rozdíl. Většina FOSS projektů má něco jako stabilní vývojovou větev. Stabilní větev by měla obsahovat dostatečně otestovaný, časem ověřený kód s minimem chyb. Na server je to určitě dobrá volba.

Často je ovšem k dispozici i testovací či vývojová verze. Ta obsahuje obvykle novější vlastnosti, které zatím ještě do stabilní verze nepronikly, neboť nebyly dostatečně otestovány. Vývojové verze obsahují zpravidla řádově více chyb než jejich stabilní protějšky, přičemž některé vlastnosti ve vývojových verzích nemusí být ještě zcela funkční (nebyly zcela implementovány), popřípadě se mohou časem i zásadně měnit – na to je třeba si dávat obzvlášť pozor, zejména při aktualizacích. Vývojovým verzím je vhodné se vyhnout, pokud to jde.

V souvislosti s vývojovými verzemi je vhodné zmínit i často používané názvosloví určující aktuální stádium vývoje: pre-alpha, alpha, beta a release candidate. Verze označená jako pre-alpha je obvykle software v raném stádiu vývoje, mnohé vlastnosti ještě nejsou vůbec implementované, mnohé se mění, často i zásadně. Taková verze by na serveru neměla co dělat, jelikož může obsahovat kritické chyby vedoucí ke ztrátě dat. Stádium alpha je na tom jen o málo lépe než pre-alpha. Mnohé vlastnosti by již měly být implementovány, i když stále je pravděpodobný výskyt kritických chyb. Ledacos také nemusí fungovat správně nebo se může časem výrazně změnit. I toto stádium bych na server důrazně nedoporučil. Stádium beta by mělo označovat dokončení všech připravovaných vlastností (ty by se již neměly měnit) a počátek testování ze strany uživatelů. Chyby jsou, přirozeně, v tomto stádiu stále očekávané. Následuje release candidate, tedy kandidát pro vydání. To už je více méně hotový produkt, ale stále se čeká na nalezení závažných chyb. Pokud nejsou nalezeny závažnější chyby během určité doby, obvykle dojde k vydání.

Poslední „verze" softwaru, se kterou můžete přijít do styku, je snapshot ze SCM nástroje (Subversion, Git, Mercurial, apod.). Je to de facto snímek zdrojového kódu platný k určitému datu – nejčastěji se bere ten nejnovější. Jelikož pokročilé nástroje pro správu verzí softwaru (SCM) umí držet současně více větví, záleží určitě, kterou větev (branch) jste si zvolili jako zdroj (záleží ovšem také na tom, jaká pravidla vývoje jsou v rámci daného projektu v platnosti). Stabilní větev by měla mít přibližně charakter stabilního vydání, vývojová větev bude v některém z výše uvedených stavů, a její nasazení na server lze proto důrazně nedoporučit. Často se ovšem s takovou verzí setkáte, jelikož v ní bude patrně opravená nějaká nepříjemná chyba (oproti poslední vývojové verzi stažitelné z webu projektu, pokud taková je).

Výše uvedená pravidla samozřejmě podléhají tomu, jak na vývoj nahlíží vývojáři projektu, který chcete nasadit. Stabilní verze jednoho projektu může stabilitou klidně odpovídat vývojové verzi jiného. Záleží na typu projektu a na vývojářích. Obecně je vhodné se nestabilním verzím vyhnout, pokud nemáte opravdu dobrý důvod je nasadit.

### Kompilace zvoleného softwaru

Software, který bude sloužit jako pokusný králík pro kompilaci, je webový server Apache, kterému se tento seriál věnoval v několika dílech. Je to pouze příklad, na kterém bude demonstrována kompilace. V žádném případě se nejedná o doporučený způsob kompilace Apache. Kromě toho, Apache určitě naleznete v repozitářích vámi používané distribuce. Budete-li chtít kompilovat právě Apache, určitě se primárně řiďte oficiální dokumentací, na kterou článek pro úplnost odkazuje.

### Nahlédnutí do dokumentace

Před tím, než budete software kompilovat, doporučuji podívat se do dokumentace, ať už dokumentace na webu projektu nebo dokumentace přibalené v příslušném zdrojovém balíčku. Obvykle zde naleznete návody či případné důležité informace týkající se kompilace ze zdrojových kódů. Uvnitř balíčku hledejte soubory README a INSTALL. V případě webového serveru Apache je dokumentace dostupná online.

### Získání a příprava zdrojového kódu

Vyberte si vhodnou verzi (dle rad výše) a stáhněte si příslušný balíček, obvykle je to tar.gz, i když tar.bz2bývá menší, tedy úspornější na přenášená data. Na serveru můžete pro samotné stažení použít nástrojwget, což se určitě hodí, máte-li na stanici, ze které se vzdáleně připojujete k serveru, pomalý upload (typické pro ADSL a mobilní připojení). Můžete si také pomoci textovým prohlížečem přímo na serveru:

```
lynx http://httpd.apache.org/download.cgi
```

Jelikož je při instalaci na server vhodné dbát zvýšené opatrnosti, je třeba si po získání balíčku ověřit kontrolní součty či digitální podpisy. Opatřete si příslušné PGP klíče a ověřte podpis náležející příslušnému souboru. Zde je samozřejmě otázka, jak si opatřit PGP klíče vývojářů z opravdu důvěryhodného zdroje. Pokud se vám nechce hledat nejbližšího vývojáře, spojit se s ním, domluvit osobní setkání, ověřit jeho občanku a zaznamenat si otisk (fingerprint) jeho klíče, můžete zkusit méně bezpečnou metodu popsanouna stránkách projektu, a sice stáhnout soubor s PGP klíči vývojářů (doporučuji alespoň přes HTTPS a přímo ze serveru apache.org), importovat tyto klíče a ověřit podpis, takto:

```
gpg --import KEYS
gpg --verify httpd-2.2.20.tar.bz2.asc
```

Po ověření balíčeк se zdrojovým kódem rozbalte:

```
tar xvf httpd-2.2.20.tar.bz2
```

### Závislosti

Závislosti, ať již kompilační či běhové (vysvětlení v minulém díle) si budete muset zajistit sami (ideálně instalací příslušných balíčků, v horším případě je budete muset zkompilovat jako první). V rámci Debianu je třeba dodat, že hlavičkové soubory potřebné pro kompilaci jsou v extra balíčcích. Pokud tedy máte jako závislost nějakou komponentu, obvykle nestačí instalovat pouze balíček s binárkou komponenty (např.libknihovna), ale musíte ještě přidat balíček s příslušnými hlavičkovými soubory pro kompilaci, tedy např.libknihovna-dev.

Nejideálnějším postupem je zjistit závislosti z dokumentace, nejméně účinným způsobem je pak spouštět skript ./configure opakovaně a orientovat se dle chybových hlášek. V Debianu však existuje ještě jeden elegantní způsob, jak závislosti řešit. Máte-li vámi kompilovaný software v repozitářích (byť v jiné verzi), můžete si ušetřit práci pomocí správce balíčků – Debian totiž umožňuje kompilaci z tzv. zdrojových balíčků, a v souvislosti s tím umožňuje jednoduchým příkazem nainstalovat všechny závislosti potřebné pro kompilaci daného balíčku, takto:

```
apt-get build-dep apache2
```

Chcete-li pouze rekompilovat balíček dostupný z repozitářů (nechcete jinou verzi), popřípadě chcete-li si sestavit vlastní balíček a hledáte vhodný vzor, můžete použít apt-src, postup je naznačen v odkazech pod článkem.

### configure

Následuje „svatá trojice", tedy ./configure, make a make install. Fáze konfigurace je ta nejnáročnější na činnost správce, jelikož se obvykle nespokojíte s pouhým spuštěním skriptu ./configure, ale budete hledat ty správné volby, které daný program zkompilují s těmi správnými vlastnostmi. Mnohé vlastnosti lze totiž zahrnout nebo naopak vyřadit vhodnými parametry. Méně vlastností znamená obvykle větší bezpečnost, naopak někdy můžete mít zájem zkompilovat daný software s parametrem, který vámi zvolená distribuce při sestavování binárního balíčku ignorovala. Zde opět odkážu na dokumentaci vámi zvoleného projektu. Dokumentace Apache je nejen v tomto ohledu vskutku vynikající, viz podrobný článek o konfiguračních volbách. Kupříkladu, pokud nechcete v Apachi používat CGI, použijete volbu --disable-cgi:

```
./configure --disable-cgi
```

Nejzásadnější volbou pro ./configure je patrně --prefix, která určuje, kam se daný software nainstaluje ve fázi make install. Specifikací konkrétního adresáře můžete klidně udržovat více verzí daného softwaru najednou:

```
./configure --prefix=/opt/apache-2.2.20
```

Pokud nespecifikujete prefix, použije se výchozí volba, která závisí na tom, co zvolil příslušný vývojář. Nejčastěji je cílem /usr/local. Já však, jak naznačuje příklad výše, preferuji /opt. To ale není důležité. Důležité je, abyste měli o souborech, které se vytvoří při instalaci, přehled, a mohli pak software snadno odinstalovat nebo spolehlivě nahradit novou verzí.

Unixový software si obvykle v adresáři specifikovaném jako prefix vytvoří vlastní strukturu s adresáři bin,lib, atd., a tam umístí příslušné soubory. V mém příkladu výše byste pak ke zkompilovanému Apachi přistupovali přes /opt/apache-2.2.20/bin/apache2ctl.

## make

Tento příkaz provede samotnou kompilaci. Máte-li vícejádrový či víceprocesorový server (nebo obojí), doporučuji přidat volbu `-j` s parametrem v podobě čísla odpovídajícímu počtu procesorů zvýšeného o jedničku. Máte-li dvouprocesorový server se čtyřmi jádry, tj. s osmi logickými procesory, zvolili byste devítku (8 jader + 1):

```
make -j9
```

V době vícejádrových a vícevláknových procesorů by byla škoda kompilovat na jednom jádře, tedy velmi, velmi pomalu v porovnání s možnostmi stroje. Na serveru, který obsluhuje klienty, je však vhodné ještě upravit prioritu kompilace. V unixových systémech se priorita nastavuje jako „niceness" (ohleduplnost), tzn. vyšší číslo znamená nižší prioritu (větší ohleduplnost k okolí). V Linuxu má priorita meze od -20 (nejvyšší priorita) do 19 (nejnižší priorita). Pokud na kompilaci nespěcháte, devatenáctka je asi nejvhodnější:

```
nice -n 19 make -j9
```

Máte-li naopak pomalý počítač, který kompiluje příliš dlouho, a na dostatečně rychlé LAN máte k dispozici silný stroj, obraťte svou pozornost k nástroji distcc, kterým můžete propojit více počítačů a vytvořit si malý kompilační „cluster".

## Instalace

Pokud kompilace proběhne bez chyb, zbývá software nainstalovat. Dosud bylo možné všechny kroky (s výjimkou používání správce balíčků) vykonávat bez práv superuživatele. K instalaci je ovšem jeho oprávnění potřeba. Výjimkou může být situace, kdy jste použili prefix k tomu, abyste software mohli nainstalovat do adresáře, který patří aktuálnímu uživateli. V takovém případě bez obav instalujte.

Pokud software nainstalujete přímo, pomocí make install, správce balíčků o těchto souborech nebude nic vědět a nebude je moci odstranit nebo vám říci, kde jsou. Můžete si samozřejmě vytvořit balíček, který nainstalujete pomocí dpkg, ale to obvykle představuje hodně práce. Existuje však i jiná možnost - nástrojcheckinstall, který se používá místo make install. Tento nástroj vám balíček na počkání balíček vytvoří:

```
checkinstall
```

Checkinstall je interaktivní program, stačí zodpovědět několik otázek, případně upravit nastavení balíčku. Poté vytvoří Checkinstall balíček obsahující váš program. Tento balíček můžete nainstalovat pomocí nástroje dpkg:

```
dpkg -i apache-custom_2.2.20-1_i386.deb
```

Checkinstall ovšem nevyplňuje závislostní vazby balíčku. Je to tedy jen poloviční řešení – správce balíčků sice bude o vašem softwaru vědět a bude ho moci odstranit, ale nebude moci v souvislosti s ním řešit závislostní vazby. Ty si budete muset ohlídat sami.

## Správa linuxového serveru: Úvod do kompilace jádra a modulů

Minulý díl probral kompilaci softwaru obecně, včetně začlenění ručně zkompilovaného softwaru do balíčkovacího systému. Dnešní díl se podívá na zoubek problematice kompilace patrně nejrozsáhlejšího softwaru, který v linuxových distribucích naleznete. Je jím samotné jádro, srdce operačního systému.

### Úvod

Na úvod opět upozorňuji na první dva díly (první díl, druhý díl) tohoto „miniseriálu" o kompilaci, které doporučuji mít přečtené, než se pustíte do čtení tohoto dílu. V případě linuxového jádra platí všechny nevýhody ruční kompilace, které byly v těchto předchozích dílech zmíněny, dokonce lze i jednu přidat – nesprávně nastavené jádro může server odstavit (systém již nenaběhne) a jedinou opravou je nabootování staršího jádra.

### Lekce z anatomie: Jádro, moduly a initrd

Jádro (kernel) je část operačního systému, která je zodpovědná za přidělování systémových prostředků procesům, hardwarovou abstrakci, správu a plánování procesů, správu paměti atd. Linux představuje nejčastěji používané jádro (a také jediné možné, má-li jít o *linuxovou* distribuci), i když existují i jiná jádra použitelná s projektem GNU. Jen tak pro zajímavost, Debian má kupříkladu k dispozici i variantu s jádrem z FreeBSD (Debian GNU/kFreeBSD) či s jádrem Hurd (Debian GNU/Hurd). V tomto seriálu se ale budu zabývat výhradně jádrem Linux.

Jádro jako takové je jedna jediná binárka, která obvykle sídlí v adresáři /boot. Často se označuje jakovmlinuz. V adresáři /boot může vedle sebe existovat mnoho verzí jader, přičemž je na zavaděči, aby konkrétní jádro po spuštění počítače zavedl do paměti a spustil. Linuxové zavaděče samozřejmě nemají problém se zaváděním různých jader podle toho, které si obsluha počítače zvolí při jeho startu.

Jádro sice může, ale také nemusí, mít veškerou funkčnost schovanou ve výše zmíněné binárce. Různé ovladače zařízení či specifické funkce mohou být zkompilovány jako moduly, které pak mohou být za běhu systému podle potřeby zavedeny nebo i odstraněny. Tyto moduly jsou k dispozici v příslušném podadresáři (jehož název odpovídá verzi použitého jádra) v /lib/modules.

Asi si teď říkáte, jak může systém naběhnout, pokud klíčový modul není součástí jádra, ale pouze jako modul. Tuto situaci nejlépe osvětlí příklad. Jádru dává zavaděč v podobě parametru informaci o tom, kde se nachází kořenový souborový systém, který má připojit. Jádro jej připojí a spustí /sbin/init, který pak zavede celý zbytek systému. Umístěním pro kořenový adresář může být třeba druhý oddíl prvního disku/dev/sda2. K tomu, aby tuto operaci jádro mohlo provést, musí ovšem umět pracovat s diskovým řadičem, ke kterému je disk připojen, a souborovým systémem, který je pro daný oddíl (/dev/sda2) použit. Co když však některý z těchto ovladačů bude zkompilován jako modul? V takovém případě by se k němu jádro nemohlo dostat a výsledkem by byl „kernel panic".

Tento problém elegantně řeší iniciální ramdisk, malý, zkomprimovaný souborový systém, uložený také v adresáři /boot (v Debianu pod názvem initrd.img s příponou v podobě verze použitého jádra), který obsahuje všechny potřebné moduly a nástroje k zavedení systému. Distribuce této možnosti hojně využívají a obsahují nástroje k jeho snadnému vygenerování. Iniciální ramdisk samozřejmě nepotřebujete, pokud vaše jádro obsahuje všechny potřebné moduly (čehož můžete dosáhnout kompilací vlastního jádra) a také pokud pro zavedení systému není třeba provést ještě něco speciálního (jako se např. dotázat uživatele na heslo šifrovaného oddílu, kde se systém nachází).

### Proč kompilovat jádro?

Pokud nepoužíváte source-based distribuci jako Gentoo, asi příliš důvodů pro odklon od distribučního jádra mít nebudete. Distribuční jádra bývají modulární, tzn. maximum funkcionality a ovladačů je dostupné v modulech, které můžete za běhu systému do jádra nahrát nebo naopak z jádra vyjmout (k tomu slouží nástroje jako lsmod a modprobe). Nemáte-li v distribučním jádru k dispozici určitý modul, není kvůli tomu třeba kompilovat celé jádro, stačí zkompilovat pouze příslušný modul (podmínkou je dostupnost zdrojového kódu jádra, který lze snadno doinstalovat pomocí správce balíčků).

Proč si tedy kompilovat vlastní jádro? Ideální je, pokud se můžete spokojit s distribučním a ušetřit si tak nejen kompilaci, ale také sledování bezpečnostních problémů a následnou rekompilaci, jakmile se objeví nějaká zranitelnost. Existují ovšem situace, kdy se vám kompilace může vyplatit.

#### *Patche*

Patch jako takový představuje úpravu zdrojového kódu jádra oproti oficiální verzi. Oficiální verze softwaru (v tomto případě jádra), která nebyla nijak upravena, se označuje jako „vanilla". Distribuční jádra téměř vždy aplikují vlastní patche, ať už to jsou úpravy funkčnosti nebo opravy chyb, které byly zpětně portovány pro stabilní verzi jádra dané distribuce.

Podstatné je, že příslušné patche je třeba aplikovat na zdrojový kód, tzn. před samotnou kompilací. Tudíž, pokud vámi zvolená distribuce zahrnula patch, který použít nechcete, nebo naopak nezahrnula patch, který použít chcete, nemáte jinou možnost než zkompilovat jádro ručně a aplikovat ty patche, které chcete.

Existuje řada úprav jádra, které přidávají nové vlastnosti nebo optimalizují jeho použití pro konkrétní účel. Některé z nich se časem dostanou do jádra a jeho vývojáři se o ně budou starat, jiným zůstává z mnoha důvodů neoficiální statut a patchování představuje jedinou možnost, jak tuto funkcionalitu dostat do vámi používaného jádra. Takovýchto patchů pro Linux existuje mnoho. Na desktopech je to např. plánovač BFS Cona Kolivase, který by měl vylepšovat odezvu a výkon na obyčejných domácích počítačích (nikoliv na serverech s mnoha procesory).

Na serveru existuje rovněž mnoho možností. Jednu kategorii tvoří bezpečnostní patche jako např.grsecurity či TOMOYO, které jsou sice neoficiální (nebo částečně neoficiální jako TOMOYO), ale mohou nabídnout jednodušší konfiguraci nebo jinou sadu vlastností než v jádře vestavěný SELinux.

Další kategorii tvoří virtualizační patche, které implementují různé druhy virtualizace (Xen, OpenVZ, VServer apod.). Možností je mnoho.

Některá jádra s příslušnými patchi (např. patche týkající se virtualizace) již distribuce mívají v repozitářích, takže rozhodně doporučuji se podívat, jestli náhodou nemáte jádro s již aplikovaným požadovaným patchem. Můžete také narazit na neoficiální repozitář s dalšími patchovanými jádry přímo pro vaši distribuci. V případě neoficiálních repozitářů je však obezřetnost určitě na místě. Nejde jenom o bezpečnost, ale také o dostatečně zodpovědný přístup správce daného repozitáře, abyste po instalaci příslušného jádra nezjistili, že dotyčný se na měsíc odmlčel, a vy máte už tři týdny v neoficiálním jádře nezáplatovanou bezpečnostní díru, která právě začíná být masivně zneužívána.

#### *Nastavení*

Nastavení vlastností jádra je druhá oblast, kvůli které se může vyplatit jádro zkompilovat. Distribuce se přirozeně snaží zvolit takové nastavení, které bude vyhovovat co nejvíce lidem, ale už z podstaty věci není možné se zavděčit všem. Některá nastavení kupříkladu sice mohou přidávat zajímavé vlastnosti, ale současně třeba snižují výkon. Správci distribucí pak mají dilema – zahrnout vlastnost, která zaujme např. deset procent správců serverů, ale všem ostatním, byť o pár procent, poklesne výkon. Podobně je to s veškerými optimalizacemi, popřípadě kompilacemi přímo pro daný typ procesoru. Distribuce volí takové nastavení, které poběží na co největším spektru hardwaru, což znemožňuje použití optimalizací pro konkrétní typ procesoru.

### Vanilla nebo distribuční kernel?

Kompilovat můžete jak distribuční jádro (zdrojáky bývají k dispozici jako samostatný balíček), tak vanilla jádro, které si stáhnete z kernel.org. Vývojáři udržují více stabilních verzí jádra, přičemž některé větve mají dokonce dlouhodobou podporu. Volba je v tomto případě samozřejmě na vás. Novější jádra obvykle vylepšují podporu hardwaru a opravují chyby, ale mohou obsahovat i regrese (což v programátorském žargonu znamená *„dříve nám to fungovalo, ale mezi tím to někdo rozbil*").

Použít distribuční verzi jádra může být v tomto směru výhodnější – jednak neobjevíte regrese (s největší pravděpodobností) a jednak se vám bude dostávat jeho oprav po dobu života distribuce.

Tím bych tento díl ukončil. Příště vám představím kompilaci jádra z praktického pohledu.

## Správa linuxového serveru: Kompilace jádra a modulů prakticky

Minule byla probrána kompilace jádra a modulů teoreticky, dnes se na ni budete moci podívat z praktického pohledu a zjistit, jak při kompilaci jádra nebo modulů postupovat.

Na úvod opět připomenu, že byste si měli projít přinejmenším minulý díl. Dozvíte se v něm mnoho podstatného a užitečného pro díl dnešní.

### Kompilace modulů

Jaderný modul představuje obvykle určitou funkcionalitu. Ve většině případů se jedná o ovladač zařízení, ale stejně tak se může jednat o modul zpřístupňující určitý souborový systém, upravující či přidávající bezpečnostní model jádra, nějakou vlastnost atd. Pokud se dostanete do situace, kdy se vám nějaký modul zalíbí, ale ve vašem distribučním jádře nebude, naštěstí nebudete muset projít kompilací jádra – postačí zkompilovat jen samotný modul a ušetřit si tak práci. Předesílám však, že bude nutné tento proces opakovat při každé změně jádra (např. po aktualizaci na novou či opravenou verzi).

Abyste mohli zkompilovat jaderný modul, potřebujete hlavičkové soubory jádra. Pokud je nemáte, musíte je nainstalovat. V Debianu se příslušný balíček jmenuje linux-headers, ale musíte vybrat ten, který patří vašemu jádru. Můžete také použít metabalík, který zajistí automatickou instalaci hlavičkových souborů, ale i těchto metabalíků je více v závislosti na architektuře, kterou využíváte. Pro 32bitové systémy bude přicházet v úvahu:

```
aptitude install linux-headers-2.6-686
```

Pro 64bitové pak:

```
aptitude install linux-headers-2.6-amd64
```

Potřebujete také nástroje pro kompilaci, popřípadě další nástroje a komponenty, které daný modul vyžaduje (dokumentace napoví). Nástroje pro kompilaci v Debianu nainstalujete prostřednictvím metabalíku build-essentail:

```
aptitude install build-essential
```

Samotný proces kompilace modulu představuje rozbalení zdrojového balíčku a jeho kompilaci:

```
tar xvf jaderny_modul.tar.gz
cd jaderny_modul
make
```

V některých případech je třeba specifikovat verzi jádra, ke které chcete modul sestavit, jinde se automaticky použije aktuálně běžící jádro. Specifikace jiného jádra má pak obvykle podobu nastavení příslušné proměnné, kterou stačí nastavit pro nástroj make, např. takto:

```
KVER=2.6.32-5-amd64 make
```

Posledním krokem je instalace modulu a aktualizace závislostí mezi moduly:

```
make install
depmod -a
```

Modul se obvykle usadí v /lib/modules/verze_jádra. Zavést jej můžete pomocí modprobe:

```
modprobe modul
```

Odstranit jej z běžícího jádra můžete příkazem:

```
modprobe -r modul
```

Chcete-li, aby se modul zaváděl při každém spuštění počítače, přidejte jeho název na prázdný řádek do souboru /etc/modules (platí pro Debian).

Proces kompilace modulu se může lišit a mít svá specifika u každého jednotlivého modulu, tudíž určitě nejprve nahlédněte do dokumentace. Samozřejmě až potom, co se podíváte do repozitářů, jestli tam už váš modul není k dispozici. V Debianu naleznete dva typy balíků příslušejících jaderným modulům, a sice balíček se zdrojovým kódem, který můžete využít ke kompilaci (takový má přídomek -source), nebo -dkmsbalíček, který se o kompilaci pro aktuální jádro postará sám, a postará se i v případě aktualizace jádra (předpokladem je dostupnost hlavičkových souborů nového jádra).

Debian také obsahuje nástroj *Module assistant* (balíček module-assistant), který usnadňuje kompilaci a instalaci řady modulů.

### Kompilace jádra

Jestli nemáte všechny nástroje potřebné pro sestavení jádra, tak si je nainstalujte:

```
aptitude install kernel-package libncurses5-dev fakeroot wget bzip2
```

Jako první je třeba, abyste si opatřili zdrojové kódy jádra. To můžete učinit buď prostřednictvím balíčkovacího systému (balíček linux-source), nebo si je stáhnout přímo ze zdroje na kernel.org. Bohužel, tento server se v době psaní článku stále zotavuje z bezpečnostního incidentu, tudíž pokud vám tato stránka momentálně nefunguje, budete si muset zdrojové kódy jádra opatřit jinde, třeba na GitHubu.

Zdrojové kódy rozbalte:

```
tar xvf linux-2.6.32.tar.gz
```

Následně (pro jistotu) zdroj vyčistěte:

```
cd linux-2.6.32
make mrproper
```

#### *Aplikace patchů*

Chcete-li aplikovat nějaký patch (pro příslušnou teorii vás odkazuji na minulý díl), nyní je vhodná doba. Patch můžete po rozbalení aplikovat nejlépe podle instrukcí na webu příslušného projektu, nebo pomocí nástroje patch, nějak takto:

```
patch -p1 < muj_patch.patch
```

Při zadávání tohoto příkazu byste se měli nacházet v adresáři s rozbaleným zdrojovým kódem jádra.

#### *Konfigurace jádra*

Dalším krokem je konfigurace jádra. Ta zahrnuje jak nastavení řady parametrů, tak určení toho, co se zakompiluje přímo do jádra a co zůstane jako modul, který lze do jádra za běhu nahrát nebo vyjmout. Tento proces je velmi komplikovaný a není vůbec těžké zapomenout na nějaký modul, bez kterého se jádro na příslušném hardwaru nerozběhne. Pokud začnete s konfigurací jádra od začátku, prozkoumejte důkladně veškerý hardware (lshw) a nahrané moduly (lsmod, modinfo), a volte příslušné moduly v nastavení jádra podle toho. Modul zakompilovaný do jádra bude fungovat, jakmile jádro naběhne. To, co bude k dispozici jako externí modul, musí přijít do iniciálního ramdisku, je-li to důležité pro zavedení systému.

Pomocnou ruku vám v této situaci poskytne aktuální konfigurace jádra, kterou můžete použít jako vzor. Konfigurace aktuálního jádra bývá k dispozici v /proc/config.gz (je-li tato funkčnost zakompilována do jádra). Toho pak můžete využít:

```
cd linux-2.6.32
cp /proc/config.gz .config.gz
gzip -d .config.gz
make oldconfig
```

V Debianu zrovna prohlížení nastavení jádra prostřednictvím /proc/config.gz dostupné není, je tedy třeba zvolit alternativní postup:

```
cd linux-2.6.32
cp /boot/config-`uname -r` ./.config
make oldconfig
```

Poslední příkaz přizpůsobí konfigurační soubor aktuálnímu jádru, což bude nejspíše zahrnovat mnoho dotazů, na které budete muset odpovědět. Poté bude konfigurace hotova a vy ji následně můžete upravit:

```
make menuconfig
```

Výše zmíněným příkazem spustíte textový konfigurační nástroj jádra. Těchto nástrojů je více, nemusíte tedy nutně používat menuconfig, můžete použít xconfig nebo jiné varianty (dostupné možnosti zjistíte pomocímake help). Pomocí těchto nástrojů vytvořte nebo upravte nastavení jádra. Až budete hotovi, uložte nastavení a opusťte nástroj.

#### *Pojmenování vámi upravené verze jádra*

Abyste odlišili svou verzi jádra od jádra, které máte nainstalované, v nastavení jádra General setup - local version vyplňte příponu, kterou chcete použít, např. -custom. Pokud tak neučiníte a jádro budete instalovat ručně bez správce balíčků, hrozí, že si přepíšete distribuční jádro!

### Sestavení jádra

Nechcete-li využívat správce balíčků, zkompilujte jádro následujícím příkazem:

```
make all
```

Následně nainstalujte zkompilované externí moduly do /lib/modules:

```
make modules_install
```

Poté zkopírujte obraz jádra do /boot:

```
cp arch/x86/boot/bzImage /boot/vmlinuz-verze_jadra
```

Zde musíte řetězec verze_jadra nahradit za celý název verze jádra (můžete se orientovat dle názvu adresáře v /lib/modules, který odpovídá vámi zkompilovanému jádru).

Následně vygenerujte iniciální ramdisk:

```
mkinitramfs -o /boot/initrd.img-verze_jadra verze_jadra
```

Zde nahraďte řetězec verze_jadra za stejný řetězec, který jste použili v příkazu pro zkopírování obrazu jádra do /boot.

Posledním krokem je úprava zavaděče:

```
update-grub
```

Budete-li chtít zužitkovat zbylé zdrojové kódy (např. pro pozdější kompilaci modulů), pusťte v adresáři s nimi toto:

```
make clean
```

Pokud jste kompilovali jádro někde bokem a ne v adresáři, kde byste chtěli zdrojáky ponechat, pak po jejich přemístění nezapomeňte upravit symbolický odkaz /lib/modules/verze_jadra/build tak, aby ukazoval na jejich nové umístění.

***Sestavení jádra v Debianu***

Debian má vlastní nástroje, které vám pomohou sestavit jádro a začlenit jej do balíčkovacího systému. Místo výše uvedeného postupu tedy můžete použít následující. Po konfiguraci jádra proveďte:

```
make-kpkg clean
fakeroot make-kpkg --initrd --append-to-version=-custom kernel_image kernel_headers
```

V příkazu výše upravte parametr --append-to-version podle toho, jakou příponu pro název vámi sestaveného jádra chcete použít. Až kompilace skončí, podívejte se do /usr/src, kde se měly vygenerovat příslušné balíčky. Ty pak nainstalujte pomocí dpkg:

```
cd /usr/src
dpkg -i linux-image-verze_jadra.deb
dpkg -i linux-headers-verze_jadra.deb
```

Iniciální ramdisk by se měl vygenerovat automaticky, totéž by mělo platit o nastavení zavaděče. Tím by měla být instalace nového jádra hotova.

## Správa linuxového serveru: Zprovoznění Ruby aplikací s RVM, Thin a Nginx

Ruby on Rails je jeden z mnoha frameworků jazyka Ruby určený pro tvorbu webových aplikací. Ačkoliv webových aplikací pro ně není tolik jako pro PHP, mnohé z nich určitě stojí za zvážení. Na rozdíl od PHP však jejich zprovoznění z pohledu správců serverů nemusí být úplně přímočaré. Tento článek osvětlí univerzální způsob nasazení Ruby (a zejména Rails) aplikací za pomoci RVM, Thin a Nginx.

### Úvod

Na úvod musím předeslat, že nejsem Rails vývojář, tudíž se na tuto problematiku dívám primárně očima správce.

Ruby on Rails není jediný framework postavený na jazyce Ruby, těch je více (viz např. tento článek nebo příslušné heslo Wikipedie) a řada z nich je velmi zajímavá. Jejich nasazení je však obvykle pro správce zvyklého na LAMP poněkud problematické, zejména v prostředí linuxových distribucí, kde se naráží na několik problémů.

Jedním kamenem úrazu je fakt, že jazyk Ruby má svůj vlastní balíčkovací systém s názvem RubyGems, který, zcela přirozeně, koliduje s balíčkovacím systémem distribucí. Je sice možné instalovat pouze balíčky z distribuce, ale pokud vaše aplikace používá konkrétní verzi, která (pochopitelně) v repozitářích není, nastává problém. Použití balíčkovacího systému Ruby pak může přinášet jiné problémy (konflikty souborů, soubory nepatřící žádnému balíčku).

Druhým kamenem úrazu je skutečnost, že jazyk Ruby zaznamenal podstatné změny (přechod z řady 1.8 na řadu 1.9) a nové verze Ruby interpretů zatím ještě nejsou podporovány řadou na Ruby postavených komponent a aplikací. V Debianu je sice možné vedle sebe nainstalovat dvě různé verze Ruby interpretu, ale např. modul Passenger pro Apache, který mu umožňuje pracovat s Ruby aplikacemi, zatím není schopen použít více než jednu verzi interpretu najednou. Není tedy možné prostřednictvím něj zprovoznit dvě aplikace, kde jedna vyžaduje starší verzi interpretu a jedna novější.

### RVM, Thin a Nginx

Výše uvedené problémy pomáhá řešit kombinace RVM a serverů Thin a Nginx. Nejprve představím jednotlivé komponenty.

RVM je zkratka pro Ruby Version Manager, tedy systém pro správu verzí Ruby. RVM můžete snadno využít k vytvoření optimálního prostředí pro běh Ruby aplikace. RVM nepotřebuje práva roota, poběží pod jakýmkoliv uživatelem a umožní vám nainstalovat jak konkrétní verzi Ruby interpretu, tak sadu „gemů" (balíčků). Dokonce je možné nainstalovat více verzí interpretu a více „gemsetů" (sad balíčků), přičemž pak se můžete mezi nimi přepínat. Má poměrně bohaté možnosti a také velmi dobrou dokumentaci.

Thin je webový server napsaný v Ruby, který by měl být rychlý, stabilní a bezpečný. Je dostupný jako Ruby „gem" (balíček).

Nginx je webový server, který velmi pravděpodobně naleznete v repozitářích vámi používané distribuce. Tento webový server má malé nároky na paměť a vysoký výkon. Umí také fungovat jako reverzní proxy, což je přesně to, co od něj v tomto řešení potřebujete. Dá se snadno propojit se serverem Thin, dokonce dvěma způsoby (přes síť a přes socket). Více informací o Nginx naleznete v jednom z předchozích dílů tohoto seriálu.

Postup řešení je následující – nejprve si vytvoříte uživatele, pod kterým bude aplikace běžet (nebo použijete nějakého existujícího). Následně zprovozníte RVM a nainstalujete potřebné verze Ruby interpretu a příslušných „gemů", které vaše aplikace vyžaduje. Poté nainstalujete gem „thin". Thin spustíte a dostanete buď sockety, nebo síťové porty, kde Thin běží. Ty pak vložíte do konfigurace Nginx a propojíte s ním konkrétní virtuální server (virtual host).

V tomto dílu popíšu obecný postup, v příštím dílu na něj navážu a prakticky jej demonstruji na zprovoznění vynikajícího FOSS systému pro správu projektů Redmine, který určitě doporučuji vaší pozornosti.

Na závěr zmíním jediný možný problém – budete-li takto nasazovat více aplikací, každou s různým Ruby interpretem a gemy, patrně budete potřebovat zajistit dostatek RAM.

### RVM

Předpokladem pro instalaci RVM je nainstalovaný Git, pokud jej nemáte, nainstalujte jej (v Debianu a podobných distribucích) příkazem:

```
aptitude install git
```

Instalace RVM je velmi jednoduchá. Nejprve však doporučuji vytvořit příslušného uživatele, pod kterým aplikace poběží.

```
adduser uzivatel
su uzivatel -
cd
```

Nyní, s právy daného uživatele, je možné provést samotnou instalaci RVM. Konkrétní podobu příkazu raději získejte z oficiální dokumentace a níže uvedený příkaz berte spíše jako ukázku (pro případ, že by se od vydání článku něco změnilo):

```
bash < <(curl -s https://rvm.beginrescueend.com/install/rvm)
```

Po instalaci by se měl automaticky upravit váš .bashrc (pokud existuje), abyste mohli RVM po přihlášení spouštět. Tím je instalace RVM jako takového hotová.

Jelikož se Ruby interpreti kompilují, bude před jejich instalací ještě třeba nainstalovat potřebné balíčky, tj. kompilátory a hlavičkové soubory. Seznam (včetně konkrétního příkazu pro instalaci) získáte zadáním:

```
rvm requirements
```

Po instalaci příslušných balíčků je možné začít s instalací konkrétní verze Ruby interpretu – zde je třeba se obrátit na dokumentaci k vámi nasazované aplikaci a zjistit, jakou verzi potřebuje. Potřebuje-li verzi 1.8.7, nainstalujete ji takto:

```
rvm install 1.8.7
```

Tuto verzi interpretu můžete pro momentální shellové sezení „aktivovat" následujícím způsobem:

```
rvm use 1.8.7
```

Pokud byste instalovali pouze jedinou aplikaci, nemusíte si dělat starosti s gemsety, tedy s možností používat různé sady Ruby balíčků s různými verzemi interpretu pro různé aplikace. Pro zjednodušení budu předpokládat právě tuto variantu. Chcete-li používat více aplikací, podívejte se do dokumentace k RVM, je velmi pěkně udělaná a vše je tam pěkně demonstrováno na příkladech.

Chcete-li učinit nějakou verzi Ruby interpretu výchozí, použijte tento příkaz:

```
rvm use 1.8.7 --default
```

Ruby aplikace si mohou s sebou tahat řadu závislostí – na vás tedy pak zbude doinstalování toho, co není distribuováno spolu s aplikací (zde je opět třeba se obrátit na dokumentaci k aplikaci). Kupříkladu, vyžaduje-li aplikace Rails framework ve verzi 2.3.11, nainstalovali byste jej takto:

```
gem install rails -v=2.3.11
```

### Instalace Thinu

Thin nainstalujete úplně stejně jako jakýkoliv gem, tj. v rámci RVM prostředí použijete příkaz:

```
gem install thin
```

### Testovací aplikace

Pro účely testování může posloužit „hello world" aplikace napsaná ve frameworku (resp. DSL) Sinatra. Vytvořte si nějaký adresář v domovském adresáři (např. testapp) a v něm soubor app.rb s následujícím obsahem:

```
require 'sinatra'

get '/' do
"LinuxEXPRES"
end
```

Následně vytvořte soubor config.ru s následujícím obsahem:

```
require './app'
run Sinatra::Application
```

Nyní můžete server v tomto adresáři spustit, a to příkazem:

```
thin -s 1 -R config.ru -a 127.0.0.1 -p 3100 start
```

Většinu parametrů je možné dekódovat užitím selského rozumu, -a udává IP, na které bude server naslouchat, -p pak konkrétní port, -R config.ru bere vámi vytvořený konfigurační soubor, který pouští Sinatra aplikaci, parametr start udává, že se má nastartovat příslušný server (funguje samozřejmě takéstop či restart). Jediný parametr, který asi není jednoduché dekódovat, je -s, který udává počet běžících serverů. V tomto případě je zvolen jeden jediný, kdyby jich bylo více, první by poslouchal na portu 3100, další na 3101 atd.

Zda se zprovoznění aplikace povedlo, můžete otestovat textovým prohlížečem Lynx:

```
lynx 127.0.0.1:3100
```

Měli byste vidět text LinuxEXPRES.

## Propojení s Nginx

Šablona pro nastavení Nginx může vypadat takto:

```
upstream  nas_thin {
server 127.0.0.1:3100;
server 127.0.0.1:3101;
server 127.0.0.1:3102;
}

server {
listen  1.2.3.4:80;
server_name  example.cz;

access_log  /var/log/nginx/example.cz.access.log;
root /home/uzivatel/testapp/public;

location / {
proxy_set_header  X-Real-IP  $remote_addr;
proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header Host $http_host;
proxy_redirect off;
if (-f $request_filename/index.html) {
rewrite (.*) $1/index.html break;
}
if (-f $request_filename.html) {
rewrite (.*) $1.html break;
}
if (!-f $request_filename) {
proxy_pass http://nas_thin;
break;
}
}

}
```

Povšimněte si direktivy `upstream` – to je jednoduchý load balancer, tedy systém pro rozložení zátěže. Využívají se tedy střídavě všechny servery. Je možné specifikovat váhy jednotlivých serverů, ale to už je mimo rozsah tohoto článku (více vám napoví dokumentace k Nginx).
Většině parametrů je asi více méně rozumět (direktiva `listen`, `server_name`, nastavení proxy, rewrite pravidla pro statický obsah). Osvětlím dvě důležité věci:

```
root /home/uzivatel/testapp/public
```

V příkladu výše nepadlo o adresáři `public` ani slovo. Webové aplikace postavené na Ruby (a nejenom ty) obvykle obsahují řadu adresářů, v nichž je zdrojový kód, konfigurace apod., a pak nějaký adresář, který se má použít jako kořen pro webovou prezentaci. Je tomu tak proto, aby se zamezil přístup k těm ostatním souborům, kde je např. konfigurace databáze apod. Budete-li nasazovat Ruby aplikace, budete nejspíše používat adresář `public` v adresářovém stromu dané aplikace.

```
if (!-f $request_filename) {
proxy_pass http://nas_thin;
break;
}
```

Toto je poslední rewrite pravidlo, které nasměruje všechny ostatní požadavky na příslušný `upstream` zdroj,`nas_thin` zde odpovídá stejnojmennému zdroji specifikovanému v rámci direktivy `upstream`.
Tím bych tento díl ukončil. V příštím díle osvětlím celý tento proces na instalaci systému Redmine.

## Správa linuxového serveru: Nasazení aplikace Redmine

Minulý díl se věnoval nasazení Ruby aplikací obecně. Tento díl vám vše ukáže na konkrétním příkladu, a sice na nasazení systému pro správu projektů Redmine.

### Redmine

Redmine je svobodný (GNU GPLv2) správce projektů, systém pro sledování chyb (bug tracker), wiki, diskusní fórum, kalendář, vše integrováno v jednom komplexním balíčku. Integrace SCM je zde samozřejmostí. Lze jej použít pro usnadnění vývoje softwaru, ale také jako aplikaci pro správu projektů obecně, nikoliv nezbytně jen těch softwarových. Umí dobře zastoupit groupware nebo zastat funkci osobního managementu. Umí dokonce i generovat Ganttův diagram. Pokud byste měli zájem o osobní management dle oblíbené metodiky GTD, Redmine vám možná nebude úplně vyhovovat – v takovém případě se můžete podívat na jinou Ruby on Rails aplikaci – Tracks. Ale zpět k Redmine. Redmine je modulární, je tedy možné vytvářet moduly rozšiřující jeho funkčnost nebo použít některé z již vytvořených modulů.

Vývoj aplikace je spravován pomocí Redmine – chcete-li se tedy podívat na běžící instanci, stačí se podívat na web projektu. Demo k vyzkoušení je pak k dispozici na subdoméně.

V tomto dílu seriálu předpokládám, že znáte obsah minulého dílu, kde najdete obecné informace o nasazování Ruby aplikací na server. Na tyto informace se budu odkazovat, resp. budu předpokládat, že je znáte. Stejně tak předpokládám, že distribucí, kterou na serveru používáte, je Debian v aktuální verzi (tj. Squeeze). V případě jiné distribuce si budete muset některé příkazy upravit.

Předesílám, že vzhledem k obsáhlosti je návod rozdělený na dva díly. Tento díl končí funkční testovací instancí Redmine, následující díl pak objasní produkční nasazení pomocí Thinu a Nginx.

### Vytvoření uživatelského účtu a zprovoznění RVM

Jako první vytvořte uživatelský účet, pod kterým bude Redmine běžet:

```
adduser redmine
```

Poté se ujistěte, že máte k dispozici SCM nástroj Git, dále nástroj curl a nástroje pro kompilaci:

```
aptitude install git curl build-essential
```

Nyní získejte oprávnění uživatele Redmine. Jste-li root, můžete provést:

```
su redmine -
cd
```

Ujistěte se, že soubor .bashrc existuje v domovském adresáři:

```
touch ~/.bashrc
```

Náhledem do dokumentace se ujistěte, že následující příkaz pro instalaci RVM je správně (mohl se změnit), a proveďte jej:

```
bash < <(curl -s https://rvm.beginrescueend.com/install/rvm)
```

Nyní byste měli mít k dispozici RVM nainstalované pod vaším redmine uživatelským účtem. RVM je třeba obvykle ještě „aktivovat" nastavením příslušných systémových proměnných, což lze provést příkazem, který vám RVM po instalaci vypíše, spolu s řadou informací, které vám doporučuji projít si. Tento příkaz si raději také ověřte:

```
[[ -s "$HOME/.rvm/scripts/rvm" ]] && source "$HOME/.rvm/scripts/rvm"
```

RVM vám doporučí instalovat pro jednotlivé Ruby interprety i určité balíčky. Opět doporučuji řídit se instrukcemi, jež vám RVM po instalaci poskytne. Uvádím nicméně seznam balíčků, který jsem obdržel já na Debianu Squeeze:

```
aptitude install build-essential openssl libreadline6 libreadline6-dev curl git-core zlib1g zlib1g-dev libssl-dev libyaml-dev libsqlite3-0 libsqlite3-dev sqlite3 libxml2-dev libxslt-dev autoconf libc6-dev ncurses-dev automake libtool bison subversion
```

Tyto balíčky je třeba nainstalovat před samotnou instalací (resp. kompilací) Ruby interpretu (viz níže), jinak vám instalace dalších komponent bude dělat problémy. V takovém případě proveďte opětovnou rekompilaci interpretu poté, co balíčky nainstalujete.

#### *Instalace Ruby interpretu a potřebných komponent*

Potřebnou verzi Ruby a příslušných komponent naleznete v přehledové tabulce v oficiálním návodu pro instalaci Redmine. V době psaní článku byl potřebný Ruby interpret pro aktuální verzi Redmine (1.2.1) buď 1.8.6, nebo 1.8.7. Vyberte si tedy a nainstalujte příslušný interpret:

```
rvm install 1.8.7
```

Tuto verzi interpretu „aktivujte" a nastavte jako výchozí:

```
rvm use 1.8.7 --default
```

Mezi další potřebné komponenty patří jednoznačně Ruby framework Ruby on Rails. Ten si ovšem Redmine tahá s sebou, pokud jej instalujete z oficiálního tarballu. Používáte-li verzi ze SCM, budete muset Ruby on Rails nainstalovat ručně. V opačném případě vám postačí nainstalovat Rack, i18n a RubyGems (viz níže):

```
gem install rack -v=1.1.1
gem install i18n -v=0.4.2
```

Nyní je třeba vyřešit problém s RubyGems, který musí odpovídat příslušné verzi Rails frameworku. Jestliže instalujete aktuální verzi (1.2.1), která vyžaduje Rails 2.3.11, pak nezapomeňte nainstalovat RubyGems verze nižší než 1.7.0:

```
rvm rubygems 1.6.0
```

V případě, že používáte starší Rails 2.3.5, musí být verze RubyGems nižší než 1.5.0.

Používáte-li Redmine ze SCM a nikoliv z oficiálního balíčku, nainstalujte Ruby on Rails v potřebné verzi. Potřebnou verzi zjistíte z přehledové tabulky, na kterou jsem vás odkázal výše. V době psaní článku to byla verze 2.3.11, kterou byste nainstalovali takto:

```
gem install rails -v=2.3.11
```

Nezapomeňte zohlednit problém s RubyGems popsaný výše a nainstalujte jeho správnou verzi, jinak při inicializaci databáze uvidíte jen chybové hlášky.

Tím je instalace potřebných komponent hotová.

### Zprovoznění databáze

Redmine umí pracovat s MySQL, PostgreSQL a SQLite. Sami zvažte, jakou databázi chcete na server nasadit (pokud vůbec nějakou). Mám zkušenost, že výchozí instalace MySQL na Debianu zabírá díky výchozímu použití InnoDB přes 100 MB RAM. Používáte-li virtuální stroj s malou RAM, můžete si pomoci vypnutím enginu InnoDB, a sice úpravou /etc/mysql/my.cnf, kam přidejte řádku s direktivou skip-innodb. PostgreSQL je ve výchozím nastavení paměťově šetrnější. Oba DBMS je však možné dodatečně vyladit a přidělit jim tolik paměti, kolik chcete nebo kolik máte k dispozici. Pro optimalizaci nastavení PostgreSQL doporučuji seriál Optimalizace PostgreSQL. Postgresu jsem se věnoval po stránce instalace a základů jeho správy i já v tomto seriálu (první díl, druhý díl).

Osobně používám Redmine na Postgresu, avšak vzhledem k tomu, že oficiální dokumentace upřednostňuje MySQL, a také vzhledem k tomu, že MySQL jsem se v tomto seriálu věnoval spíše méně, v tomto návodu upřednostním MySQL. Ujistěte se tedy, že máte nainstalovaný MySQL server:

```
aptitude install mysql-server
```

Pokud jej budete instalovat takto z balíčku, budete vyzváni k zadání rootovského hesla pro MySQL. I když je možné ponechat heslo prázdné, rozhodně jej **důrazně doporučuji nastavit**.

Pro lepší a rychlejší interakci s databází MySQL je třeba nainstalovat příslušný Ruby balíček pro práci s MySQL. Ještě před tím, než se do jeho instalace pustíte, nainstalujte hlavičkové soubory MySQL klienta (bez toho se kompilace příslušného gemu nezdaří):

```
aptitude install libmysqlclient-dev
```

Poté jako uživatel redmine proveďte instalaci gemu MySQL:

```
gem install mysql
```

Chcete-li použít jiný databázový server, nahlédněte do instalační příručky Redmine, jsou tam podrobné informace, jak databázi nastavit.

#### *Vytvoření databáze a databázového uživatele*

Spusťte MySQL konzoli a přihlaste se jako root:

```
mysql -u root -p
```

Následně vytvořte databázi s kódováním UTF-8 a českým porovnáváním:

```
CREATE DATABASE `redmine` DEFAULT CHARACTER SET utf8 COLLATE utf8_czech_ci;
```

Vytvořte uživatele s příslušným (pokud možno silným) heslem:

```
CREATE USER 'redmine'@'localhost' IDENTIFIED BY 'heslo';
```

A konečně, přidělte uživateli všechna práva pro tuto databázi:

```
GRANT ALL PRIVILEGES ON `redmine`.* TO 'redmine'@'localhost';
```

## Instalace Redmine

Stáhněte si aktuální verzi Redmine a přeneste ji na server (nebo použijte textový prohlížeč Lynx či wget přímo na serveru). Balíček následně rozbalte a vstupte do nově vytvořeného adresáře s aplikací:

```
tar xvf redmine-1.2.1.tar.gz
cd redmine-1.2.1
```

Nyní je třeba Redmine propojit s vámi používanou databází. Vytvořte tedy konfigurační soubor config/database.yml s následujícím obsahem (kde samozřejmě upravíte název databáze, uživatelské jméno a heslo):

```
production:
adapter: mysql
database: redmine
host: localhost
port: 3306
username: redmine
password: heslo
```

Můžete také použít šablonu config/database.yml.example a tu upravit.

Nyní vygenerujte „session store secret":

```
rake generate_session_store
```

A konečně, inicializujte databázi a nechte vytvořit příslušné tabulky v rámci databázové migrace:

```
RAILS_ENV=production rake db:migrate
```

Objeví-li se chyby, zkontrolujte práva k databázi. Následující krok sice není povinný, ale je důrazně doporučený, jelikož databázi naplní výchozími daty, jako jsou role, oprávnění apod. (bez tohoto kroku budete mít opravdu „holý" Redmine):

```
RAILS_ENV=production rake redmine:load_default_data
```

V tuto chvíli můžete konečně ověřit funkčnost instalace Redmine spuštěním webového serveru Webrick:

```
ruby script/server webrick -e production
```

Server by se měl spustit na portu 3000 a konzolový výpis by měl vypadat takto:

```
=> Booting WEBrick
=> Rails 2.3.11 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2011-10-15 16:40:50] INFO  WEBrick 1.3.1
[2011-10-15 16:40:50] INFO  ruby 1.8.7 (2011-06-30) [i686-linux]
[2011-10-15 16:40:50] INFO  WEBrick::HTTPServer#start: pid=32709 port=3000
```

Webový server běží na všech přidělených IP adresách, takže můžete Redmine vyzkoušet, jestli se nezobrazí jen „bílá obrazovka smrti". Pokud Redmine běží, server shoďte a postupujte dál. Pokud ne, všimněte si chybové hlášky a pokuste se problém vyřešit. Zejména si znovu projděte výše uvedené instrukce, zda jste provedli opravdu vše, zda databáze funguje apod.

Pokud byste se chtěli přihlásit, vězte, že výchozí uživatel je „admin" a heslo je (překvapivě) také „admin". Jako správcům vám, doufám, nemusím zdůrazňovat, abyste přinejmenším heslo urychleně změnili.

Tímto bych tento díl ukončil. Příště se budete moci dozvědět, jak nainstalovat Thin a Nginx a připravit je pro produkční nasazení Redmine na externí IP adrese.

## Správa linuxového serveru: Redmine: Thin a Nginx

Minulý díl načal nasazení systému pro správu projektů Redmine. Dozvěděli jste se, jak dospět k funkční testovací instanci Redmine. V tomto dílu se dočtete, jak nasadit Redmine produkčně pomocí webových serverů Thin a Nginx.

### Úvod

Tento návod je dokončením, které navazuje na předchozí dva díly seriálu. Doporučuji vám si tyto dva díly (1. díl, 2. díl) projít, pokud jste tak ještě neučinili.

### Webový server Thin

Minulý díl skončil funkční instancí Redmine, kterou jste si mohli otestovat pomocí webového serveru Webrick, jenž se ovšem nehodí pro produkční nasazení. Dalším krokem je tedy zprovoznění webového serveru Thin. Jak bylo řečeno v prvním dílu miniseriálu o nasazení Ruby aplikací, Thin je server napsaný v Ruby, zralý pro produkční nasazení (na rozdíl od Webricku, který se doporučuje používat pouze na testování Ruby aplikací). Thin lze propojit s Nginx, což provedeme za chvíli. Získejte tedy oprávnění uživatele Redmine:

```
su redmine -
cd
```

A následně nainstalujte Thin:

```
gem install thin
```

Pro samotné spouštění Thin můžete použít vlastní skript, který Thin nahodí (popř. podle potřeby shodí). Pokud budete používat pokročilejší vlastnosti RVM, můžete specifikovat konkrétní gemset před spuštěním Thinu. Pokud jste postupovali podle návodu, nemusíte si s touto informací dělat starosti.

Vytvořte adresář $HOME/bin:

```
su redmine -
cd
mkdir bin
```

V tomto adresáři vytvořte skript pojmenovaný redmine-thin:

```
nano bin/redmine-thin
```

Obsah skriptu může být následující:

```
#!/bin/bash

if [ -z "$1" ]; then
echo 'Specifikujte parametr pro Thin server (napr. "start", "stop" apod.)'
exit 0
fi

[[ -s "$HOME/.rvm/scripts/rvm" ]] && source "$HOME/.rvm/scripts/rvm"
cd "$HOME/redmine-1.2.1"
thin -e production $1 -s1 --socket "$HOME/redmine.sock" -d
```

Tento skript si vylaďte podle libosti. Zejména parametry Thinu, jako např. počet serverů (parametr -s) či konkrétní soubor se socketem, přes který bude Thin komunikovat. Socket je rychlejší než komunikace přes síť, proto je lepší používat socket – v případě problémů se můžete vrátit k použití sítě a spustit Thin na nějakém portu na místní smyčce (viz minulý díl). Dodávám jen, že Thin mění pojmenování socketu, i když je nastaven pro běh jen jednoho serveru, přidává tam číslo (redmine.0.sock). Zohledněte to pak v konfiguraci Nginx. Následně učiňte skript spustitelným:

```
chmod a+x bin/redmine-thin
```

Adresář $HOME/bin můžete zařadit do prohledávací cesty, tedy do proměnné PATH. Do .bashrc uživateleredmine tedy přidejte:

```
PATH="$PATH:$HOME/bin"
```

Spusťte Thin pomocí skriptu:

```
$HOME/bin/redmine-thin start
```

Následně zkontrolujte, zda byl příslušný socket vytvořen (pozor, chvilku to trvá, zejména na pomalejších serverech, nelekejte se tedy, že se tam soubor hned neobjeví). Pokud ne, podívejte se v adresáři s Redmine do logu, který je umístěn v log/thin.0.log, zda vám nenapoví, kde je problém. Tento soubor pak můžete spouštět s právy roota takto:

```
su redmine -c '/home/redmine/bin/redmine-thin start'
```

Aby se tento skript spustil při startu serveru, nejjednodušší je přidat příkaz k jeho spuštění do/etc/rc.local, ale mnohem čistší řešení je vytvořit pro něj skript v /etc/init.d a zařadit ho do příslušné úrovně běhu pomocí update-rc.d. Abyste zajistili jeho nepřerušený běh, resp. jeho případný automatický restart, pokud by spadl, můžete použít nástroj Monit. Tento nástroj byl představen ve dvou dílech seriálu (1. díl, 2. díl).

### Nginx

Nyní by vám již měl Redmine běžet a měl by být dostupný na socketu /home/redmine/redmine.sock. Zbývá propojit webový server s tímto socketem. Nainstalujte tedy Nginx:

```
aptitude install nginx
```

Nginx je třeba nakonfigurovat. Ideální je, pokud vám Redmine bude běžet přes zabezpečený protokol HTTPS, k čemuž ovšem potřebujete certifikát, a to ideálně ne self-signed (na který jsou prohlížeče v dnešní době již silně alergické), ale certifikát podepsaný od nějaké důvěryhodné certifikační autority. Nechcete-li příliš utrácet, můžete použít např. StartSSL nebo CACert.org, které vám vydají certifikát zdarma.

Vytvořte tedy konfigurační soubor pro Nginx:

```
nano /etc/nginx/sites-available/redmine
```

A naplňte jej následujícím obsahem:

```
upstream  backend {
server    unix:/home/redmine/redmine.0.sock;
}

server {
listen  1.2.3.4:443;
server_name  jmeno.serveru.cz;

client_max_body_size 10M;
client_body_buffer_size 128k;

ssl  on;
ssl_certificate  /etc/nginx/certifikat.pem;
ssl_certificate_key  /etc/nginx/klic.pem;

ssl_session_cache  builtin:1000  shared:SSL:10m;
ssl_session_timeout  3m;

access_log  /var/log/nginx/redmine.access.log;
root /home/redmine/redmine-1.2.1/public;

location / {
proxy_set_header  X-Real-IP  $remote_addr;
proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header Host $http_host;
proxy_redirect off;
```

```
if (-f $request_filename/index.html) {
rewrite (.*) $1/index.html break;
}
if (-f $request_filename.html) {
rewrite (.*) $1.html break;
}
if (!-f $request_filename) {
proxy_pass http://backend;
break;
}
}

}

server {
listen 1.2.3.4:80;
rewrite ^(.*) https://$host$1 permanent;
}
```

Pokud byste přeci jen nechtěli používat zabezpečený přístup ke své instanci Redmine a stačil by vám nešifrovaný protokol HTTP, můžete použít následující konfigurační soubor:

```
upstream  backend {
server   unix:/home/redmine/redmine.0.sock;
}

server {
listen  1.2.3.4:80;
server_name  jmeno.serveru.cz;

client_max_body_size 10M;
client_body_buffer_size 128k;

access_log  /var/log/nginx/redmine.access.log;
root /home/redmine/redmine-1.2.1/public;

location / {
proxy_set_header  X-Real-IP  $remote_addr;
proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header Host $http_host;
proxy_redirect off;
if (-f $request_filename/index.html) {
rewrite (.*) $1/index.html break;
}
if (-f $request_filename.html) {
rewrite (.*) $1.html break;
}
if (!-f $request_filename) {
proxy_pass http://backend;
break;
}
}

}
```

V tomto souboru si určitě upravte všechny potřebné informace jako IP adresy serveru, dále cesty k logům, cestu k Redmine (direktiva root), cestu k socketu nebo socketům, samozřejmě pak také cesty k certifikátům v SSL variantě konfiguračního souboru výše apod. Budete-li používat více než jeden běžící server, socketů se vytvoří také více – v takovém případě ke všem z nich uveďte cestu:

```
upstream  backend {
server   unix:/home/redmine/redmine.0.sock;
server   unix:/home/redmine/redmine.1.sock;
server   unix:/home/redmine/redmine.2.sock;
server   unix:/home/redmine/redmine.3.sock;
}
```

Posledním krokem je "aktivace" konfigurace, resp. vytvoření symbolického odkazu v adresáři/etc/nginx/sites-enabled:

```
ln -s /etc/nginx/sites-available/redmine /etc/nginx/sites-enabled/redmine
```

A konečně můžete spustit nakonfigurovaný Nginx:

```
/etc/init.d/nginx start
```

Nyní se přihlaste do Redmine a konfigurujte, vytvářejte projekty, uživatelské účty apod. Připomínám, že výchozím uživatelem je "admin" a heslo k němu je "admin". Opět dodávám, že by bylo vhodné toto heslo urychleně změnit.

**Doladění konfigurace Redmine**

Na závěr zbývá doladit konfiguraci Redmine, a sice vyřešit rozesílání e-mailů. Redmine má velmi propracovaný systém e-mailových oznámení, a proto potřebuje přístup k poštovnímu serveru, aby mohl poštu rozesílat. Asi nejjednodušší je použít nástroj sendmail k odesílání pošty, ale můžete použít i klasický SMTP server. Získejte tedy oprávnění uživatele redmine a dostaňte se do adresáře s Redmine:

```
su redmine -
cd ~/redmine-1.2.1
```

Můžete použít předlohu, configuration.yml.example, a zkopírovat ji do neexistujícího configuration.yml:

```
cp config/configuration.yml.example config/configuration.yml
```

Tento soubor je bohatě komentovaný, varianta se sendmailem je tam naznačena:

```
production:
email_delivery:
delivery_method: :sendmail
```

Stejně tak varianta s použitím místního poštovního serveru:

```
production:
email_delivery:
delivery_method: :smtp
smtp_settings:
address: "localhost"
port: 25
```

K dispozici jsou i varianty s externím poštovním serverem s autentifikací. Pokud budete používat tento soubor jako předlohu, berte na vědomí, že je v něm již doručování e-mailů nastaveno (SMTP server s autentifikací), je tedy vhodné tuto konfiguraci najít a upravit (soubor je velký a klíčový odkomentovaný kousíček snadno přehlédnete). Tím je instalace a konfigurace Redmine hotová. Z bezpečnostního hlediska běží Thin a celé Redmine pod separátním, neprivilegovaným účtem redmine. Ke Thinu se pak připojuje Nginx, který také běží pod neprivilegovaným účtem, v Debianu pod účtem www-data. S MySQL, Thinem a Nginx jsem se na OpenVZ virtuálním stroji s 256 RAM vešel do 150 MB, i s poštovním serverem. Použil jsem jeden paralelní Thin server, jak jsem naznačil v návodu výše.

## Správa linuxového serveru: Git na vlastním serveru

Systémy pro správu verzí jako Git nacházejí široká uplatnění, nejenom v rámci programování. V tomto článku se dozvíte něco málo o Gitu, zejména jak vytvořit centrální úložiště pro Git repozitář na vlastním serveru.

### Úvod

Systém pro správu verzí (známý pod zkratkami SCM nebo VCS) je software, který spravuje změny ve vašich (zejména textových) datech. Můžete tak snadno zjistit, ve kterých souborech se co změnilo, kdy se to změnilo a kdo to změnil. Pomocí SCM se organizují práce vývojářů na softwarových projektech. Ale SCM poslouží na jakákoliv textová data, nejen na zdrojový kód. V unixových systémech to často bývají konfigurační soubory, které mají v těchto systémech textový charakter. Mohou to ale také být třeba zdrojové kódy rozsáhlejšího dokumentu tvořeného v LaTeXu či za pomoci konverzních nástrojů typu Asciidoc, Pandoc, txt2tags apod.

Git byl vyvinut pro potřeby vývoje linuxového jádra (kde nahradil komerční a nesvobodný Bitkeeper) a postupem času se stal jedním z nejpopulárnějších SCM nástrojů, alespoň ve světě svobodného softwaru.

Existuje řada služeb, které vám umožní vytvořit si svoje vlastní Git repozitáře, hostované na serveru příslušného poskytovatele. GitHub je zde nejznámějším, ale nikoliv jediným příkladem. Tyto služby však mohou být za určitých okolností zpoplatněné (např. když nechcete svůj repozitář vystavit světu) a samozřejmě podléhají určitým podmínkám, které se vám mohou, ale také nemusí líbit. Máte-li vlastní server, můžete si centrální úložiště pro Git repozitář(e) vytvořit na něm.

Pokud máte ještě v živé paměti články o zprovoznění Redmine na linuxovém serveru, dodám, že Redmine podporuje řadu SCM nástrojů a podpora Gitu samozřejmě nechybí.

### Distribuovaný SCM a centrální úložiště

Git je distribuovaný SCM, což znamená, že nepotřebuje centrální bod, se kterým by musel udržovat spojení, aby byl schopen realizovat základní úkony (tím se liší od centralizovaných SCM jako CVS či SVN). Všechny základní operace prováděné Gitem jsou realizovány lokálně, v rámci vašeho souborového systému (konkrétně pak v podadresáři .git). Proto jsou také pekelně rychlé. Centrální server tedy v zásadě není nutný, ale má-li na projektu pracovat více lidí, nebo chcete-li online zálohu či centrální bod, abyste mohli snadno synchronizovat své změny napříč několika počítači, centrální server se vám může hodit.

Je třeba zmínit, že Git není jediným pod Linuxem dostupným distribuovaným SCM. Můžete použít např. Mercurial nebo Bazaar.

### Základy práce s Gitem

Na Git jako takový se zde podrobněji zaměřovat nebudu, ale jakousi základní kuchařku vám přesto poskytnu.

Je-li to poprvé, co vůbec používáte Git, bylo by dobré se mu představit (ideálně ještě před prováděním commitů):

```
git config --global user.name 'Jméno Příjmení'
git config --global user.email mail@example.cz
```

Vytvoření Git repozitáře představuje jeden jediný příkaz, a to v adresáři, který chcete Gitem spravovat (samozřejmě včetně podadresářů):

```
git init
```

Tento příkaz vytvoří výše zmíněný .git podadresář s příslušnými výchozími konfiguračními soubory. Poté je třeba přidat příslušné soubory:

```
git add .
```

Výše uvedený příkaz přidá celý aktuální adresář včetně podadresářů. Můžete samozřejmě specifikovat konkrétní soubor nebo soubory:

```
git add soubor1.txt soubor2.txt
```

Následně můžete vytvořit první commit, tedy uložit aktuální stav sledovaných souborů do repozitáře:

```
git commit -m 'Popis commitu'
```

V tuto chvíli by měl být aktuální stav všech sledovaných souborů uložen. Jakmile provedete změny, můžete je uložit, tedy „commitnout“ buď automaticky (commitnou se všechny změněné soubory):

```
git commit -a -m 'Popis commitu'
```

Nebo můžete selektivně „commitnout“ jen něco:

```
git add soubor.txt
git commit -m 'Popis změny souboru soubor.txt'
```

Aktuální změny oproti poslednímu commitu je možné zjistit pomocí:

```
git status
```

Podrobnější přehled o změnách můžete zjistit pomocí:

```
git diff --stat
```

A konkrétní diff pak obdržíte pomocí:

```
git diff
```

Samotný git add ukládá daný soubor do tzv. staging area, tudíž pokud mezi tím příslušný soubor změníte a pak provedete commit, tak uložíte změny, které byly aktuální v době zadání git add. Soubory uložené ve „staging area“ můžete vyjmout pomocí git reset HEAD.

Existují také grafické nadstavby Gitu, které vám usnadní získání informací o změnách, popřípadě samotnou správu repozitáře, například gitk (měl by být součástí distribuce Gitu).

Chcete-li se o používání Gitu dozvědět více, podívejte se do závěru článku, kde je řada odkazů na zdroje o Gitu, a to i v češtině. Nechybí ani jedna velice užitečná kniha přeložená do češtiny.

### Git přes SSH

V tomto dílu vám představím to nejjednodušší řešení centralizovaného Git repozitáře, které existuje. Na serveru vám k tomu stačí pouze běžící SSH server. Toto řešení je vhodné pro soukromé repozitáře, které nechcete nikde vystavovat a kde nepotřebujete přístup více osob. Na komplikovanější řešení se zaměřím v dalších dílech.

Nacházíte-li se v adresáři s Git repozitářem, jako první krok exportujte repozitář do správného „formátu“ a překopírujte jej na server. Předpokladem je, že v domovském adresáři vzdáleného uživatele uzivatelexistuje adresář git.

```
git clone --bare .git /tmp/muj_repositar.git
scp -r /tmp/muj_repositar.git uzivatel@server.example.cz:/home/uzivatel/git/muj_repositar.git
```

Nyní je repozitář na serveru vytvořen. Stačí tedy Gitu říci, že tento vzdálený repozitář existuje:

```
git remote add origin ssh://uzivatel@server.example.cz/home/uzivatel/git/muj_repositar.git
```

V tuto chvíli máte možnost commitnuté změny poslat na server, popřípadě si změny z centrálního repozitáře stáhnout, a samozřejmě máte možnost vzdálený repozitář naklonovat třeba na nový počítač. Samotné publikování změn na server provedete takto:

```
git push
```

Máte-li na serveru dostupné změny, které nemáte k dispozici v lokálním repozitáři, můžete si je stáhnout a začlenit:

```
git pull
```

No a konečně, pokud se ocitnete na novém počítači, kde daný repozitář není k dispozici, můžete jeho aktuální stav naklonovat pomocí git clone, tímto způsobem:

```
git clone ssh://root@lucifer.krkavec.net/root/git/muj_repositar.git
```

Toto řešení je velmi jednoduché, ale v zásadě pokrývá pouze ty nejprostší potřeby. Máte-li více lidí a potřebujete-li definovat k repozitáři třeba read-only přístup, potřebujete mírně sofistikovanější řešení, kterým se budu zabývat v příštím dílu seriálu.

## Správa linuxového serveru: Pokročilejší nasazení Gitu

Minulý díl se věnoval představení SCM nástroje Git a vytvoření centrálního úložiště na vlastním serveru. Řešení představené minule bylo to nejjednodušší možné. V tomto dílu zmíním jeden tip pro efektivnější používání Gitu, dále vám ukážu, jak repozitáře zveřejnit pro read-only přístup, a nakonec vám povím, jak obsah repozitářů prohlížet přes webové rozhraní.

### Úvod

Pokud přicházíte poprvé k problematice SCM nástrojů, tedy nástrojů pro správu verzí, měli byste si nejprve přečíst minulý díl, kde byl Git představen.

### Základy práce s Gitem: Ignorování souborů

Git není navržen k verzování jednoho jediného souboru. Verzuje se vždy celý adresář a o veškerých souborech, které v něm existují, ale nejsou verzovány, dostanete hlášení po každém „git status“:

```
# On branch master
# Changed but not updated:
#   (use "git add ..." to update what will be committed)
#   (use "git checkout -- ..." to discard changes in working directory)
#
#       modified:   prace.txt
#
# Untracked files:
#   (use "git add ..." to include in what will be committed)
#
#       prace.txt~
no changes added to commit (use "git add" and/or "git commit -a")
```

Tyto soubory mohou představovat dočasné soubory, záložní kopie editoru (jako prace.txt~ v příkladu výše), logy, zkrátka vše, co sice v adresáři zůstane, ale v repozitáři to nechcete. Abyste udržovali výpis čistý a přehledný, je vhodné Gitu říci, které typy souborů má ignorovat. Ty se pak ve výpisu změn nebudou zobrazovat

Řešení jsou dvě. Obě zahrnují soubor s definicí vzorů jmen souborů, které se mají ignorovat, např.:

```
*~
*.pdf
!zachovat.pdf
*.aux
*.[ab]
```

Příklad obsahu souboru by měl být samovysvětlující. Nebudou se brát v úvahu soubory se jmény končící na ~, dále všechny PDF soubory s výjimkou souboru zachovat.pdf, který ignorován nebude, dále všechny soubory .aux a všechny soubory s příponami .a nebo .b.

Tento soubor je třeba vhodně pojmenovat a umístit. Pokud chcete, aby byl součástí repozitáře (a verzoval se taky), vytvořte v příslušném adresáři soubor .gitignore. Chcete-li tyto vzory držet mimo repozitář, využijte soubor .git/info/exclude (cesta je relativní ke kořenovému adresáři repozitáře).

### Veřejný read-only přístup k repozitáři pomocí Git démona

Jednou z možností, jak zpřístupnit veřejnosti Git repozitář pro read-only přístup (tedy pro klonování repozitáře a běžnou synchronizaci, nikoliv však pro zápis), je využít samotného Git démona a přistupovat k repozitáři prostřednictvím Git protokolu. V Debianu je třeba za tímto účelem nainstalovat balíček git-daemon:

```
aptitude install git-daemon
```

Git démon v Debianu pracuje s repozitáři umístěnými ve /var/cache/git, kde je samozřejmě nejspíše nemáte. Je tedy třeba si dopomoci symbolickými odkazy. Dávám zde k dispozici dva příklady podle typu umístění repozitáře, vyberte si ten, který odpovídá vašemu repozitáři:

```
ln -s /cesta/k/repositar/.git /var/cache/git/repositar.git
ln -s /cesta/k/repositar.git /var/cache/git/repositar.git
```

Pokud se v tuto chvíli pokusíte repozitář naklonovat, obdržíte suchou chybovou hlášku:

```
fatal: The remote end hung up unexpectedly
```

Kdybyste někdy řešili problém, proč se toto děje, podívejte se na server do /var/log/git-daemon/current, dozvíte se konkrétní příčinu. V tomto případě ohlásí Git chybu „repository not exported“, což znamená, že danému repozitáři nebyl povolen export Git démonem. Proto je třeba ještě vytvořit následující soubor, kterým export povolíte:

```
touch /var/cache/git/repositar.git/git-daemon-export-ok
```

Repozitář pak můžete naklonovat tímto způsobem:

```
git clone git://server.example.org/repo.git
```

### Prohlížení Git repozitáře online pomocí nástroje Gitweb

Gitweb je nástroj pro prohlížení Git repozitářů prostřednictvím webového prohlížeče. Můžete si procházet historii, stahovat snapshoty, prohlížet změny atd., a samozřejmě to také umožnit návštěvníkům vašeho webu. Nástroj si můžete prohlédnout v akci na git.kernel.org, na některém z repozitářů obsahujícím linuxové jádro.

Instalace Gitwebu je poměrně jednoduchá, stačí nainstalovat příslušný balíček:

```
aptitude install gitweb
```

Máte-li nainstalovaný webový server Apache, Gitweb by měl začít fungovat okamžitě – vytvoří se alias/gitweb, tudíž zadejte URL následujícího typu: http://server.example.org/gitweb. Konfiguraci Gitwebu naleznete v /etc/apache2/conf.d/gitweb. Pokud tedy chcete změnit daný alias nebo si Gitweb zpřístupnit jen na určitém virtuálním serveru (virtual host), soubor upravte nebo přesuňte jinam a jeho obsah vložte tam, kde ho mít chcete. Připomínám, že webovému serveru Apache jsem se v tomto seriálu již věnoval, a sice ve třech dílech (v prvním, druhém a třetím).

Používáte-li jiný webový server, budete muset upravit jeho nastavení pro Gitweb. Aplikaci Gitweb naleznete v /usr/share/gitweb, je napsaná v Perlu a patrně budete využívat CGI nebo FastCGI. Návod pro zprovoznění Gitwebu pomocí webového serveru Nginx naleznete v odkazech pod článkem.

Gitweb má konfigurační soubor umístěný v /etc/gitweb.conf, je poměrně strohý, ale přesto obsahuje pár věcí, které můžete chtít změnit, jako logo, odkaz na každé stránce (vedoucí třeba zpět na váš web) atd.

Pokud jste svoje repozitáře nepojmenovali, asi vás příliš neuspokojí pohled na popisek *„Unnamed repository; edit this file 'description' to name the repository.“*, zde stačí udělat přesně to, co popisek říká, upravit soubor .git/description v daném repozitáři.

Musím vás upozornit ještě na jednu věc. Zatímco Git démon vyžaduje výše zmíněný git-daemon-export-ok, Gitweb zpřístupní vše, co se nachází v nastaveném adresáři (výchozí je /var/cache/git), bez ohledu na tuto značku. Chcete-li upravit, které repozitáře bude Gitweb zpřístupňovat a které ne, máte dvě možnosti. Buď v /etc/gitweb.conf změníte proměnnou $projectroot na nějaký jiný adresář, a do něj vložíte příslušné symbolické odkazy těch repozitářů, které zveřejnit chcete. Nebo vytvoříte seznam projektů, což je soubor obsahující položky ve formátu cesta_k_projektu vlastnik@example.com a cestu k tomuto souboru vložíte jako hodnotu proměnné $projects_list v /etc/gitweb.conf, takto:

```
$projects_list = "/var/cache/git/projekty.txt";
```

Soubor s projekty (repozitáři) může mít následující obsah:

```
repositar1.git vlastnik1@example.org
repositar2.git vlastnik2@example.org
```

Cesta k repozitářům může být relativní nebo absolutní, v příkladu výše je cesta relativní (resp. cesta vypadá neuvedená, jelikož soubor se seznamem projektů je ve stejném adresáři jako repozitáře).

Tím bych tento díl ukončil. Příště vám představím řešení centrálního repozitáře s jemnějším systémem oprávnění než „plný přístup nebo žádný přístup“, které bylo prezentováno v minulém díle.

## Správa linuxového serveru: Víceuživatelský Git server s Gitosis

Minulý díl byl věnován zpřístupnění Git repozitářů přes webové rozhraní pomocí webové aplikace Gitweb a také přes Git démona pro read-only přístup. V tomto dílu se dozvíte, jak zpřístupnit Git repozitáře více lidem a definovat pravidla přístupu pomocí nástroje Gitosis.

### Úvod

Pokud jste nečetli poslední dva díly, které se věnovaly SCM nástroji Git, základům jeho používání a jeho jednoduchému nasazení na vlastní server, projděte si první díl a druhý díl, abyste byli v obraze.

Představili jsme zde jednoduché techniky zprovoznění centrálního Git repozitáře. Probrali jsme v zásadě dvě možnosti – veřejný read-only přístup k repozitáři prostřednictvím Git démona a kompletní (read-write) přístup přes SSH. Obojí je velmi jednoduché na zprovoznění (je-li vůbec třeba něco zprovozňovat), avšak pro některá použití nemusí takto jednoduché techniky představovat optimální řešení, jelikož jim chybí možnost jemněji definovat pravidla přístupu a hlavně neumožňují bezpečný přístup více uživatelů k jednomu repozitáři. Gitosis je jedním z nástrojů, který umožňuje přístup k repozitářům definovat detailněji a pro více uživatelů. Lze jej integrovat s webovou aplikací Gitweb a Git démonem, i když to v Debianu nefunguje out-of-the-box (postup, jak toho docílit, bude naznačen).

Dodám, že Gitosis neumožňuje úplně nejjemnější dělení přístupových práv. V zásadě umožňuje víceuživatelský přístup k jednotlivým repozitářům a pro každého uživatele (nebo skupinu) umožňuje definovat oprávnění buď k read-only nebo read-write přístupu. Ačkoliv je to oproti dříve představeným metodám podstatné zlepšení, existují ještě sofistikovanější nástroje, jako např. Gitolite.

Gitosis používá ke svému nastavení Git. Při instalaci dojde k vytvoření správcovského repozitáře, prostřednictvím kterého je pak možné měnit nastavení a připravovat půdu pro nové repozitáře. Pro přístup je používán SSH server a veřejné klíče. Pokud vám není jasná problematika použití SSH klíčů, přečtěte si můj Úvod do SSH pro správce. Ve zbytku článku předpokládám, že tuto problematiku ovládáte.

Gitosis je vázán na jednoho konkrétního uživatele, v Debianu je to stejnojmenný uživatel, tedy gitosis. Gitosis řídí přístup jednotlivých uživatelů prostřednictvím konfigurace a SSH klíčů. Žádné uživatelské účty se při práci s ním nezakládají. Z pohledu systému vše probíhá pod jedním uživatelem.

### Instalace a zprovoznění Gitosis

Prvním krokem nemůže být nic jiného než nainstalování příslušného balíčku. V repozitářích Debianu je k dispozici stejnojmenný balíček gitosis, který nainstalujte takto:

```
aptitude install gitosis
```

Pokud se vás během instalace Debconf nezeptá na SSH klíč, který má být použit pro přístup k repozitáři správce, nedojde k vytvoření tohoto repozitáře, a bude tedy nutné jej vytvořit ručně (obvyklý trik s dpkg-reconfigure gitosis zde nejspíše nezabere, alespoň dle mých pokusů). Ruční vytvoření repozitáře je však velmi jednoduché:

```
sudo -H -u gitosis gitosis-init < id_rsa.pub
```

Předpokladem je dostupnost souboru id_rsa.pub s veřejným SSH klíčem, který budete používat pro přístup ke správcovskému repozitáři gitosis-admin.

#### *Nastavení Gitosis*

V tuto chvíli byste měli mít možnost naklonovat správcovský repozitář na svém počítači.

```
git clone gitosis@server.example.org:gitosis-admin.git
```

Pokud se podíváte do obsahu repozitáře, naleznete soubor gitosis.conf (hlavní konfigurační soubor Gitosis) a adresář keydir. Ten slouží jako úložiště SSH klíčů jednotlivých uživatelů.

Výchozí konfigurační soubor je velmi strohý. Příklad komentovaného konfiguračního souboru pro inspiraci naleznete např. v /usr/share/doc/gitosis/examples/example.conf (platí pro Debian).

Obecné nastavení Gitosis naleznete v bloku [gitosis]:

```
[gitosis]
gitweb = no
daemon = no
```

Zde se nastavuje globální zpřístupnění repozitářů Gitwebu a Git démona (aby toto nastavení mělo kýžený účinek, je třeba provést integraci těchto dvou nástrojů s Gitosis, což je naznačeno níže). Zpřístupnění repozitářů těmto nástrojům lze nastavit jak na globální úrovni zde, tak na úrovni jednotlivých repozitářů (viz níže).

#### *Nastavení repozitáře*

Nastavení repozitáře není nutné, stačí jméno repozitáře přiřadit konkrétní skupině uživatelů (viz níže). Pokud však chcete nastavit některé parametry repozitáře, vytvořte nový blok s jeho definicí, takto:

```
[repo mujprojekt]
gitweb = yes
description = Popis mého projektu
owner = Michal Dočekal
daemon = yes
```

V definici je naznačená integrace s Gitwebem a Git démonem (viz výše). Přístupnost repozitáře přes Gitweb řídí proměnná gitweb, přičemž pokud ji nastavíte na yes, doporučuji vyplnit i proměnné description (popis) a owner (jméno vlastníka). Tyto dvě proměnné slouží primárně pro integraci s Gitwebem. Poslední proměnná, daemon, řídí integraci s Git démonem.

#### *Integrace s Gitwebem*

Za účelem integrace Gitosis s Gitwebem upravte hodnoty v konfiguračním souboru /etc/gitweb.conf, takto:

```
$projectroot = "/srv/gitosis/git";
$projects_list = "/srv/gitosis/gitosis/projects.list";
$strict_export = "true";
```

#### *Integrace s Git démonem*

Integrace s Git démonem je trošku složitější. V první řadě je potřeba upravit parametry startovacího skriptu Git démona v souboru /etc/service/git-daemon/run. Zde nalezněte řádku se specifikací --base-path:

```
--base-path=/var/cache /var/cache/git
```

Jak je vidět, tato cesta neukazuje do úložiště Gitosis, tudíž v tomto stavu Git démon jednotlivé repozitáře „neuvidí". Nastavte tedy správnou cestu, takto:

```
--base-path=/srv/gitosis/git /srv/gitosis/git
```

Soubor uložte a následně restartujte Git démona:

```
sv restart git-daemon
```

Poté by povolené repozitáře mělo být možné klonovat, takto:

```
git clone git://server.example.org/mujprojekt1.git
```

#### *Nastavení uživatelů a přístupových práv*

Výchozí nastavení Gitosis zahrnuje skupinu gitosis-admin, jejíž členové mají práva k repozitáři správce:

```
[group gitosis-admin]
writable = gitosis-admin
members = docekal
```

Pokud byste chtěli vytvořit skupinu skupina obsahující uživatele milan a jan s přístupem k repozitářůmmujprojekt1, mujprojekt2 a mujprojekt3 tak, aby uživatelé skupiny měli přístup k repozitářům mujprojekt1 amujprojekt2 pouze pro čtení a k repozitáři mujprojekt3 i se zápisem, vypadala by konfigurace takto:

```
[group skupina]
writable = mujprojekt3
readonly = mujprojekt1 mujprojekt2
members = milan jan
```

Mezi členy (members) může patřit i jiná skupina – tu je ovšem třeba označit zavináčem na začátku:

```
members = @nejaka_skupina
```

U uživatelů se předpokládá existence souborů s jejich SSH klíči. Uživatel `milan` by měl mít klíč `vkeydir/milan.pub`. Po nastavení a přidání klíčů změny commitněte a nahrajte na server:

```
git add gitosis.conf keydir
git commit -m 'nastaveni repositaru'
git push
```

**Workflow pro jednotlivé repozitáře**

Ve chvíli, kdy máte hotové nastavení oprávnění a repozitářů, je konečně můžete začít tvořit a nahrávat na server. Sada příkazů od vytvoření repozitáře až k jeho prvotnímu nahrání na server by mohla vypadat takto:

```
mkdir projekt
cd projekt
git init
git remote add origin gitosis@server.example.org:projekt.git
# vytvoření souborů a adresářů s obsahem repozitáře
git add *
git commit -m "initial commit"
git push origin master:refs/heads/master
```

Význam většiny příkazů by měl být patrný. Jádro postupu tvoří dva příkazy. První slouží k přidání serveru s Gitosis jako vzdáleného repozitáře:

```
git remote add origin gitosis@server.example.org:projekt.git
```

Druhý pak v prázdném repozitáři na serveru vytvoří „master“ větev z místní „master“ větve. Příkaz nemusíte opakovat, podruhé stačí git push.

```
git push origin master:refs/heads/master
```

Tím bych tento díl ukončil.

## Správa linuxového serveru: Víceuživatelský Git server s Gitolite

Minulý díl představil nástroj Gitosis, který umí zprostředkovat víceuživatelský přístup k centralizovanému Git repozitáři. Gitolite pracuje podobně jako Gitosis, avšak umožňuje ještě jemnější pravidla přístupu. Více se dozvíte v článku.

### Úvod

Na úvod vás musím upozornit na poslední tři díly tohoto seriálu, které se věnovaly SCM nástroji Git, základům jeho používání a jeho nasazení na vlastní server, počínaje těmi nejjednoduššími způsoby (OpenSSH) až po víceuživatelské řešení s Gitosis v minulém díle. Projd si první, druhý a třetí díl, abyste byli v obraze. Gitolite představuje z řešení zde probíraných nejsofistikovanější nástroj pro víceuživatelský přístup k centralizovanému Git repozitáři. Oproti Gitosis umí omezit právo zápisu na konkrétní větve (branches), respektive reference (refs). Můžete kontrolovat i to, zda má uživatel právo provést „rewind", tedy „odčinění" commitů nebo jiné změny historie. Je třeba si dát pozor na jednu věc. Gitolite neumí omezit read-only přístup na úroveň jednotlivých větví. Konkrétnímu uživateli tedy můžete zakázat read-only přístup k celému repozitáři, ale nikoliv ke konkrétním větvím. Je to dáno tím, že Git neumí rozlišit mezi jednotlivými větvemi, pokud provádí čtení. Tento problém spolu s alternativami řešení je podrobně popsán v dokumentaci.

### Instalace

Instalace Gitolite v Debianu je jako už tradičně velmi jednoduchá, neboť stačí nainstalovat příslušný balíček z oficiálních repozitářů:

```
aptitude install gitolite
```

Pokud při instalaci nedojde k vytvoření správcovského repozitáře, což se alespoň v mém případě nestalo, je třeba jej inicializovat ručně. Naštěstí je k tomu připraven nástroj gl-setup. Ten musíte spustit s právy uživatele gitolite, pod kterým bude Gitolite pracovat. Tento nástroj vyžaduje jeden parametr, a tím je cesta k souboru s veřejným SSH klíčem, který bude využíván pro přístup ke správcovskému repozitáři. Tady pozor – jelikož tento nástroj musí běžet s právy uživatele gitolite, je třeba, aby k umístění souboru s klíčem měl tento uživatel přístup, čehož lze docílit např. zkopírováním souboru s veřejným SSH klíčem do/tmp. Následně stačí provést:

```
su gitolite -c 'gl-setup /tmp/uzivatel.pub'
```

Uživatelé Gitosis mají uživatelská jména shodná se jménem souboru s veřejným SSH klíčem. Proto je vhodné při provádění příkazu výše jméno souboru přizpůsobit, abyste pak neskončili s uživatelem "id_rsa". Vše jde však samozřejmě později upravit.

### Základní konfigurace

V tuto chvíli byste měli být schopní naklonovat konfigurační repozitář:

```
git clone gitolite@server.example.org:gitolite-admin
```

Repozitář by měl obsahovat dva adresáře: adresář conf s konfiguračním souborem (gitolite.conf) a adresář keydir obsahující soubory s SSH klíči. Přidání nového uživatele tedy vyžaduje přidání jeho SSH klíče do adresáře keydir v podobě souboru se jménem ve tvaru uzivatel.pub a následné přidání příslušných práv v conf/gitolite.conf.

### Uživatelé a skupiny

Skupiny se od uživatelů liší v tom, že jsou uvozeny zavináčem. Výchozí skupinou je skupina @all, zahrnující všechny uživatele. Ostatní skupiny můžete snadno nadefinovat:

```
@vyvojari = michal ludek jana
@spravci = root admin
@personal = @vyvojari @spravci
```

Skupina se samozřejmě může skládat z více skupin (viz definice skupiny @personal) a uživatelé mohou patřit do různých skupin. V systému oprávnění je možné konkrétní právo přiřadit uživateli, skupině, nebo obojímu.

### Systém oprávnění

Základní práva, která můžete přiřadit jednotlivým uživatelům a skupinám, mají následující podobu:

| Oprávnění | Význam |
|---|---|
| R | read-only přístup |
| RW | právo provést push do existující větve a vytvořit novou větev |
| RW+ | totéž co RW, avšak navíc právo přepisovat, resp. ničit (push -f) |
| - | odepřít přístup |

Příklad definice nového repozitáře tedy může vypadat takto:

```
repo  projekt
    RW+  =  michal @spravci
    RW   =  ludek jana
    R    =  pepa
```

Podstatnou změnou oproti Gitosisu není však pouze zjemnění práv, ale také možnost příslušná práva aplikovat na konkrétní větve, resp. na konkrétní reference (refs). Názorněji to bude vypadat na příkladu:

```
repo  projekt
    RW  master$           =  michal @spravci
    RW  refs/heads/master$  =  michal @spravci
```

Mezi oprávněním a rovnítkem může být regulární výraz, který definuje konkrétní referenci, k níž má dotyčný daný přístup. Neuvedete-li na začátku výrazu refs/, doplní si Gitolite interně na začátek refs/heads/ – oba zápisy oprávnění v příkladu výše jsou tedy ekvivalentní. Znalci regulárních výrazů určitě tuší, že dolar na konci značí konec řádku, tj. výše uvedený zápis povolí uživateli michal zápis do větve master, ale už ne do větve nazvané např. mastermind. Pokud by tam dolar na konci nebyl, měl by právo zápisu i k větvimastermind. Pokud vám regulární výrazy nic neříkají, bylo by dobré se na ně podívat. Pod článkem naleznete několik odkazů, které by vám měly tuto problematiku osvětlit. Pokud mezi oprávněním a rovnítkem nic neuvedete, bude se příslušné oprávnění vztahovat na celý repozitář se všemi větvemi. Podobně lze zacházet i s tagy – je možné uživatelům povolit nebo nepovolit přiřazovat tagy jako takové nebo tagy v určitém tvaru.

```
repo projekt
    RW  refs/tags/rc[0-9] = michal
    -   refs/tags/rc[0-9] = ludek
    RW  refs/tags         = ludek
```

Zde je uživateli michal uděleno právo vytvářet tagy ve tvaru rc[číslo], je tedy možné vytvořit tag rc1, rc10, ale také rc1a. Uživatel ludek má oprávnění vytvářet libovolně pojmenované tagy, ovšem s výjimkou tagů začínajících na rc[číslo]. Budete-li zacházet s oprávněním pro odepření přístupu (mínusem), mějte na paměti, že záleží na pořadí prováděných pravidel. Kdybyste příklad výše otočili a dali prostřední řádek na konec, uživatel ludek by mohl vytvářet jakékoliv tagy a přístup by mu nikdy odepřen nebyl. Tolik k základům konfigurace Gitolite. Mnohé pokročilé vlastnosti Gitolite zde nezazněly, a (nejen) proto vám doporučuji podívat se na oficiální dokumentaci ke Gitolite, která je velmi podrobná a přívětivá, čtivě psaná a plná užitečných příkladů. Uživatelům Gitolite důrazně doporučuji si ji projít.

### Workflow

Styl práce s Gitolite je velice podobný práci s Gitosis. Jakmile vytvoříte vhodnou půdu pro nový repozitář úpravou konfiguračních souborů, jejich commitem a nahráním (push) na server, stačí u místního repozitáře přidat vzdálený server:

```
git remote add origin gitolite@server.example.org:repositar.git
```

A následně provést push na server:

```
git push origin master:refs/heads/master
```

Tím by mělo dojít k vytvoření „master" větve v repozitáři na serveru. Na závěr zmíním ještě jeden tip – pokud se na server přihlásíte přes SSH, oznámí vám Gitolite, k čemu máte jaký přístup (Gitosis toto neumí):

```
ssh gitolite@server.example.org
hello admin, the gitolite version here is 1.5.4-2+squeeze1 (Debian)
the gitolite config gives you the following access:
R  W        gitolite-admin
```

```
R  W        projekt
@R @W       testing
```
Tím bych tento díl ukončil.

## Správa linuxového serveru: Git hooky a alternativy ke Githubu

V tomto dílu bude problematika Gitu uzavřena. Budou představeny Git hooky, tedy možnost skriptování navázaného na události v Git repozitáři a alternativy ke Githubu, které můžete nasadit na svůj server.

### Git hooky

Git obsahuje jednu velmi podstatnou vlastnost, která se dá velmi dobře využít na serveru. Pokud jste třeba někdy zabrousili do vod frameworku Ruby on Rails, určitě jste slyšeli o hostingu Heroku, kde se nasazení (deployment) aplikací řeší právě přes Git. Je jistě velmi pohodlné provést aktualizaci webové aplikace na serveru prostřednictvím obyčejného git push. Takovou funkcionalitu je možné realizovat právě pomocí Git „háčků". Git hook je obyčejný shellový skript, který je umístěn v podadresáři hooks v adresáři obsahujícím Git repozitář (obvykle .git/hooks). Git tyto hooky spouští v reakci na jisté události, resp. v konkrétním okamžiku (před či po nějaké operaci). To, zda bude skript spuštěn, nebo ne, je dáno jeho oprávněním (je-li spustitelný, bude spuštěn). Ke tvorbě skriptu můžete využít jakéhokoliv skriptovacího jazyka, nebo zůstat u Bashe (popř. jiného oblíbeného shellového interpretu).

Seznam jednotlivých hooků i s jejich případnými parametry a činností, na kterou je příslušný hook, naleznete v tabulce níže:

| Název skriptu | Vázáno na | Parametry |
|---|---|---|
| applypatch-msg | běh git-am skriptu | název souboru s popisem commitu |
| pre-applypatch | běh git-am, po aplikování patche a commitu | - |
| pre-commit | běh git-commit před commitem | - |
| prepare-commit-msg | běh git-commit ihned po připravení výchozího popisu commitu | 1-3 parametry: jméno souboru s popisem commitu, zdroj zprávy (viz dokumentace), SHA1 commitu |
| commit-msg | běh git-commit | název souboru s popisem commitu |
| post-commit | běh git-commit po provedení commitu | - |
| pre-rebase | běh git-rebase před prováděním rebase | - |
| post-checkout | běh git-checkout po aktualizaci pracovního stromu | 3 parametry: ref předchozí HEAD, ref následující HEAD, druh checkoutu (viz dokumentace) |
| post-merge | běh git-merge po úspěšném provedení merge | typ merge (viz dokumentace) |
| pre-receive | běh git-receive-pack na serveru před prováděním aktualizace | žádné, ale přijímá vstup na STDIN (viz dokumentace) |
| update | běh git-receive-pack na serveru po aktualizaci dat, ale před aktualizací referencí | název reference, starý SHA1, nový SHA1 |
| post-receive | běh git-receive-pack na serveru po aktualizaci referencí | žádné, ale přijímá vstup na STDIN (viz dokumentace) |
| post-update | běh git-receive-pack na serveru po aktualizaci referencí | proměnlivý počet, seznam aktualizovaných referencí |
| pre-auto-gc | běh git-gc --auto, před provedením operace | - |

Přehledovou tabulku doporučuji doplnit dokumentací. Stejně tak doporučuji si projít příklady skriptů, které by měly být součástí libovolného Git repozitáře. Naleznete tam nejenom příklady použití, ale také užitečné komentáře o použití daných háčků.

Stejně jako existují místní a vzdálené Git repozitáře, můžete specifikovat různé místní a vzdálené háčky. Můžete tedy používat jednu sadu funkcionality lokálně a jinou vzdáleně.

Jak je z přehledové tabulky asi patrné, jednotlivé háčky je možné použít pro ledacos. K výše zmíněné aktualizaci webové aplikace po provedení „push" operace z místního repozitáře může posloužit háček post-update. K samotnému rozbalení obsahu repozitáře můžete použít třeba tento postup:

```
git archive --format=tar HEAD | (cd /var/www/prezentace && tar xf -)
```

Na Git háčky je ale možné „pověsit" i samotnou správu repozitářů, různé kontroly a pojistky. Ukončení skriptu jiným chybovým stavem než nulou (např. exit 1) má v některých případech za následek selhání dané operace. Můžete tak odmítat push na základě specifických kritérií.

### Alternativy ke Githubu

Github je nepochybně velmi zdařilý projektový hosting. Není však jediným hostingem podporujícím Git. Ten podporuje i řada jiných, jako např. Sourceforge, Google code apod. To vše jsou však webové služby. Nejsou open source a není možné je nasadit na vlastní server. Máte možnost buď akceptovat jejich podmínky, nebo se poohlédnout po nějaké alternativě. K těmto projektům existuje několik alternativ.

#### *Girocco*

Girocco je Git hosting, který lze označit jako GitWeb na steroidech. Girocco je na Gitwebu založeno a ovládá se stejně. Obsahuje samozřejmě potřebné vlastnosti pro Git hosting, jako registraci uživatelů/projektů apod., a některé vlastnosti navíc. Autorem je Petr Baudiš a software je provozován na známém repozitáři repo.or.cz. Pro svou funkci potřebuje webový server, Git a GitWeb. Lze ho propojit s Git démonem. Podporuje zrcadlení repozitářů (můžete jej tedy používat jako záložní úložiště pro některé své repozitáře hostované jinde) a forkování projektů.

#### *Gitorious*

Gitorious je založený na Ruby on Rails a šířený pod licencí GNU Affero GPL verze 3. Ta se liší od klasické GNU/GPL tím, že zdrojový kód je třeba zpřístupnit nejen příjemcům při distribuci softwaru, ale také při jeho zpřístupnění uživatelům přes síť. Gitorious je Web 2.0 aplikace ve stylu Githubu, je mu tedy o něco blíže než Girocco. Oproti GitWebu obsahuje navíc vestavěnou wiki, kterou je možné použít pro dokumentaci projektu. Žádný z těchto projektů neobsahuje všechny funkce Githubu, jako např. komentování, bug tracker apod. Pokud takové funkce potřebujete, je možné zkombinovat některé z těchto řešení s jiným open-source projektem – kupříkladu, open-source bug trackerů existuje celá řada (Flyspray, Bugzilla, Mantis apod.). Jediné, co budete muset oželet, je integrace těchto nástrojů s Git hostingem.

#### *Redmine, Trac a ostatní*

Kromě Git hostingů existují také systémy pro správu (softwarových) projektů, které mohou mít rovněž podporu Gitu. Ta může zahrnovat jak prohlížení repozitářů, tak třeba integraci s bug trackerem. Vytvoření repozitáře a přístup (se zápisem) je obvykle nutné řešit jinak, nicméně tyto nástroje mohou obsahovat řadu užitečných komponent jako integrované wiki, bug trackery apod. Mezi tyto nástroje patří v tomto seriálu již zmiňovaný Redmine nebo např. Trac.

### Závěr

Tím bych tento díl, stejně jako téma Git uzavřel. Na závěr nemohu nezmínit, že existuje řada jiných SCM nástrojů, kterým jsem se zde nevěnoval (např. Mercurial, Bazaar, SVN apod.). Řada z nich je open source a je možné (v případě centralizovaných SCM i nutné) je nasadit na server. Jsou opatřeny různými vlastnostmi a rozhodně bych doporučoval se na ně podívat.

## Správa linuxového serveru: Tor na serveru: Úvod

Tor je anonymizační nástroj, který nabízí různé typy použití jak na desktopu, tak na serveru. V tomto dílu bude představen Tor jako takový, dále možnosti jeho nasazení na server a hlavně hrozby, se kterými se při jeho používání můžete setkat.

### Úvod

Tor je, jak už bylo poznamenáno výše, anonymizační nástroj. Jeho název je zkratkou pro „The onion router", tedy „cibulový" router. Zjednodušeně, pracuje tak, že náhodně vybere několik uzlů z Tor sítě a vytvoří z nich „obvod" (circuit), přes který posílá šifrovaně veškerou komunikaci. Každý z uzlů tuší pouze, ze kterého uzlu a na který další uzel data tečou. Nedokáží odhadnout původního odesílatele a ani příjemce. Díky šifrování také netuší, jaká data vlastně přepravují.

Způsob šifrování se inspiroval cibulí - jedná se o „vrstvené" šifrování. Jednotlivé uzly vždy svým klíčem paket dešifrují a odešlou dalšímu uzlu, který proceduru opakuje. Konečným bodem komunikace je výstupní uzel (exit node), kde se paket dešifruje úplně (odloupne se poslední vrstva „cibule") a odešle. Odpověď na něj se pak pošle sítí zpět. Obvod jako takový se samozřejmě čas od času mění, takže nová spojení pak chodí jinudy.

Tor má však jednu vlastnost navíc. Dokáže anonymizovat nejenom komunikaci směrem „ven", tedy do internetu, ale také komunikaci směrem „dovnitř". Server připojený k Tor síti může anonymizovat některou svou běžící službu a nabídnout ji členům Tor sítě. Tato „vnitřní" síť se běžně nazývá „darknet".

Na anonymizaci se nelze spoléhat 100%, což jednak osvětlí hrozby uvedené níže, a jednak to říká i samotný Tor, který při startu vypisuje hlášku:

This is experimental software. Do not rely on it for strong anonymity.

Tato hláška říká, že byste měli považovat Tor za experimentální software a nespoléhat na něj jako na poskytovatele silné anonymity. Jinými slovy, určitě je třeba zvážit, jakým způsobem a k čemu budete Tor využívat, stejně jako rizika z toho vyplývající.

Na závěr úvodu je třeba zmínit, že Tor není jediná služba podobného typu. Anonymizačních služeb a sítí existuje samozřejmě více.

### Možnosti nasazení Toru na server

Tor je svobodný software napsaný v jazyce C a šířený pod BSD licencí. Je možné jej nasadit nejenom jako klienta, který vám umožní brouzdat po webu anonymně nebo objevovat temná zákoutí darknetu, ale je také možné jej nasadit i na server, a to hned čtyřmi způsoby.

První možností je vytvořit si **Tor relay**, tedy „zapojit" váš server do sítě Tor, aby přes něj mohla téct data uživatelů a anonymita sítě jako takové mírně vzrostla. Zde máte dvě možnosti - buď vytvoříte pouze průchozí uzel, přes který budou data téct, ale který nebude sloužit jako výstupní uzel (**exit relay**), nebo vytvoříte uzel, který bude sloužit i jako výstupní.

Seznam Tor uzlů je veřejný - pokud se tedy připojíte k síti Tor (jako relay), zařadí se váš server automaticky do tohoto seznamu. To vyplývá ze samotného pricipu sítě - klient potřebuje v první řadě seznam uzlů, aby byl schopen vytvořit obvod. Pokud se vám toto nezamlouvá, stále můžete vytvořit „tichý" uzel (**bridge relay**), který se nikde zveřejňovat nebude. Díky tichým uzlům pak může Tor využívat i uživatel připojený přes ISP, který Tor jako takový blokuje.

Poslední variantou je možnost anonymizovat některou službu, která běží na vašem serveru (webový server, apod.), a vytvořit tzv. **hidden service** (skrytou službu). Klienti Tor sítě pak k této službě budou moci přistupovat, aniž by tušili, kde se server nachází, nebo aniž by váš server měl šanci zjistit, kým je vlastně využíván. Pro tuto možnost využití Toru nepotřebujete, aby váš server pracoval jako relay.

### Hrozby spojené s Torem

Jelikož Tor je jeden z nejpoužívanějších anonymizačních nástrojů, čelí celé řadě hrozeb. Ačkoliv se uživatelské stránce věci nechci příliš věnovat vzhledem k povaze seriálu, přesto si dovolím hlavní body hrozeb použití Toru rozebrat, a to včetně hrozeb z pohledu uživatele (některé z nich jsou relevantní i pro správce serverů, zejména pak správce nějaké „skryté služby").

#### *Hrozby z pohledu uživatele*

Zásadní problémy Toru jsou dvě věci. Tou první je fakt, že Tor neřeší anonymitu samotných aplikací. To znamená, že Tor může fungovat bezchybně, ale aplikace (např. webový prohlížeč) ochotně prozradí vaši skutečnou IP adresu (za tímto účelem lze v případě webového prohlížeče použít skriptovací jazyky a „rozšiřující" technologie jako Java, JavaScript, Flash, apod.). Může také docházet k „prosakování" citlivých informací jako např. názvy navštívených webů, které bude špatně nastavená (nebo špatně napsaná) aplikace získávat z místního DNS resolveru, který se bude nejspíše ptát DNS serverů vašeho poskytovatele.

Druhým zásadním problémem je vztah anonymity a bezpečnosti. Tor řeší anonymitu (v rámci možností), ale neřeší bezpečnost. Používání Toru vás jednoznačně vystavuje hrozbám spojeným s výstupními uzly. Ty totiž mají kontrolu nad vaší komunikací. Mohou vás šmírovat, zaznamenávat všechna hesla posílaná v plain textu (což se v minulosti dělo a může se kdykoliv dít znovu) a mohou realizovat všemožné MITM (man-in-the-middle) útoky.

Očividnou hrozbou je pak samotné chování uživatele, jako např. podepsání příspěvku nebo specifický styl psaní, který je možné porovnat s dřívějšími příspěvky, a dopátrat se vaší IP adresy z doby, kdy jste daný web navštěvovali bez použití Toru.

Trošku pokročilejší hrozbou je pak korelační analýza, která může na základě časových a jiných údajů identifikovat uživatele, má-li útočník pod kontrolou vstupní a výstupní uzel (nebo přístup k datům, která k nim tečou).

Samotné prohlížeče dnes představují velký problém, jelikož o sobě prozrazují příliš mnoho informací. Unikátní nebo méně obvyklé nastavení prohlížeče vás může identifikovat nebo vaší identifikaci napomoci. Jedná se o tzv. „browser fingerprinting". Z tohoto důvodu se tedy často nedoporučuje používat stejný prohlížeč pro normální brouzdání webem a pro brouzdání prostřednictvím Toru.

Řada webů dnes využívá komponenty od Googlu či od Facebooku, čímž těmto společnostem dává možnost analyzovat pohyb uživatelů po webu. Navštěvujete-li tedy stejné stránky stejným způsobem (nebo hůř - se stejným prohlížečem) bez ohledu na použití Toru, je možné, že některá z těchto firem si vaši identitu propojí.

#### *Hrozby z pohledu správce serveru*

Osobně vidím z pohledu správce dva problémy při použití Toru. První je možnost zařazení Tor uzlu do nejrůznějších blacklistů. To může být způsobeno jednak tím, co se přes váš server posílá (máte-li uzel s povoleným výstupem), a jednak jistým „proaktivním" přístupem některých subjektů, které plošně blokují nebo zařazují do blacklistů IP adresy Tor uzlů bez ohledu na to, jestli se dané počítače něčeho skutečně dopustí nebo nikoliv.

Nebývá neobvyklé, že se zakáže uživatelům přístup nebo možnost přispívání, pokud přicházejí ze sítě Tor. Horší je to pak s blacklisty, které jsou používány na úrovni zpracování pošty pro boj se spamem. Pokud tedy na stejné IP adrese provozujete poštovní server, můžete se na nějaký z blacklistů dostat a mít pak problém s doručováním pošty.

Druhým možným problémem může být reakce policie na nějaký incident, který zapříčinil uživatel Toru prostřednictvím vašeho serveru (týká se samozřejmě výstupních uzlů). Nejedná se pouze o to, že se můžete stát předmětem vyšetřování (jelikož je to váš server), ale také o to, co se stane s příslušným serverem a oblastí jeho uložení. Stále mám v živé paměti zátah, při kterém slovenští policisté zabavili 10 serverů jisté webhostingové společnosti, a úspěšně tak „odpojili" 3500 webů (souvislost s Torem to nemělo, ale problém to jistě ilustruje). Určitě si tedy rozmyslete, kde budete případný výstupní uzel provozovat, aby případný zátah a konfiskace serveru nevyřadila důležité služby. Stejně tak důrazně doporučuji informovat dotčeného poskytovatele (serverhousingu, VPS, zaměstnavatele, apod.).

Jelikož řada poskytovatelů (serverhousingu, VPS, apod.) dnes vydává podmínky, ve kterých činí zákazníky odpovědné za škody, které vzniknou v souvislosti s jimi používanou službou (bez ohledu na to, zda-li ji způsobil klient, útočník nebo někdo úplně jiný), doporučuji si znovu pročíst příslušné podmínky a rozmyslet se, popřípadě to s příslušným subjektem probrat i v této rovině.

### Závěr

Tor je nástrojem, který poskytuje uživatelům jistou míru anonymity, stejně jako možnost obejít vládní či korporátní cenzuru internetu. Podobně jako jakýkoliv jiný nástroj je možné Tor využít jak k etickému konání, tak k neetickému (potažmo také k nezákonnému).

Jak už jsem zmínil v úvodu, způsob použití Toru je třeba zvážit. Zde zmíněný výčet hrozeb jistě není úplný, ať už proto, že jsem na něco zapomněl já, nebo tím, že ne všechny hrozby byly dosud objeveny.

Tím bych tento úvodní díl ukončil. V příštím díle vám Tor představím z technického hlediska.

## Správa linuxového serveru: Skrytý server pomocí Tor

Minulý díl představil anonymizační nástroj Tor. Tento díl osvětlí jeho nasazení v podobě skrytého serveru, tj. postup, jakým je možné anonymizovat poskytovanou službu.

### Úvod

Pokud jste nečetli minulý díl, doporučuji vám si jej projít, jelikož obsahuje základní informace o Toru, jeho fungování i jeho úskalích. Skrytá služba (hidden service) je služba běžící na vašem serveru, anonymizovaná (a dostupná) prostřednictvím Toru. Obecně se může jednat o jakoukoliv službu, nemusí to nutně být zrovna webový server, i když ten to bývá nejčastěji. Váš server získá určitý název v podobě „domény" druhého řádu s TLD.onion, která funguje pouze v rámci Toru. Prostřednictvím tohoto názvu se pak budou k vašemu serveru moci připojit klienti. Jak konkrétně skryté služby fungují, se můžete dozvědět na webu Toru. Jako úplně první krok, ještě před samotnou anonymizací nějaké služby, bych doporučoval zvážit, čeho se snažíte dosáhnout, zda je použití Toru skutečně vhodné, jaké jsou jeho nevýhody a slabiny apod. Můžete samozřejmě prostřednictvím Toru zpřístupnit nějakou všeobecně dostupnou službu jako svůj osobní blog (pak vás případná ztráta anonymity nemusí trápit), nebo vytvořit nějakou službu dostupnou jen prostřednictvím Toru.Když už jsem se dotkl ztráty anonymity, doporučuji se také pečlivě zamyslet nad tím, jaký dopad by na vás tato ztráta měla, a zařídit se podle toho. Nechci se vzhledem k minulému dílu příliš opakovat, ale tvůrci Toru jej v současné době stále považují za experimentální software, u kterého byste se neměli vyloženě spoléhat na zachování anonymity. Více viz minulý díl seriálu.

### Volba softwaru pro skrytý server

Máte-li představu, jakou službu budete anonymizovat, zůstává zásadní otázkou, jaký konkrétní software byste měli použít. Osobně doporučuji použít takový, který klade důraz na bezpečnost a který dobře znáte (a tudíž jej umíte dobře nastavit).To platí pro veškerý nasazovaný software. Mějte na paměti, že nasazení blogu na platformě Wordpress představuje nasazení nejenom webového serveru, ale i PHP interpretu, databáze MySQL a samotného Wordpressu. Zabezpečení a pravidelné aktualizace je pak třeba vztáhnout na všechny tyto komponenty.

### Instalace a nastavení Toru

Samotná instalace Toru je velmi jednoduchá – postačí nainstalovat balíček s názvem tor, což lze v Debianu provést takto:

```
aptitude install tor
```

Toru trvá nějakou dobu, než sestaví funkční okruh. Na rozdíl od většiny ostatních služeb tedy nezačne fungovat okamžitě, ale až po nějaké době. Průběh můžete sledovat v logu (/var/log/tor/log). Až se Toru podaří sestavit okruh, uvidíte tyto hlášky:

```
[notice] Tor has successfully opened a circuit. Looks like client functionality is working.
[notice] Bootstrapped 100%: Done.
```

Takto by však Tor fungoval pouze jako klient. Samotné vytvoření skryté služby vyžaduje úpravu konfigurace Toru, která je uložena v souboru /etc/tor/torrc. Definice skryté služby může vypadat takto:

```
HiddenServiceDir /var/lib/tor/sluzba1/
HiddenServicePort 80 127.0.0.1:80

HiddenServiceDir /var/lib/tor/sluzba2/
HiddenServicePort 80 127.0.0.2:80
HiddenServicePort 22 127.0.0.2:22
```

V tomto případě je naznačeno, jak vytvořit více než jednu službu a více než jeden port. Jádro pro nastavení skryté služby představují dva kroky:

- určení adresáře pro uložení důležitých informací o službě, jako např. soukromý klíč (volba `HiddenServiceDir`)
- určení cílového (kde bude služba nabízena klientům přes Tor) a zdrojového (kde služba běží) portu pro nabízenou službu (volba `HiddenServicePort`) – těch může následovat více

Pokračovat byste měli restartem Toru:

```
/etc/init.d/tor restart
```

Adresáře ani soukromé klíče není třeba vytvářet, vše bude vytvořeno automaticky po restartu Toru, včetně vygenerování jména serveru – to pak může vypadat třeba takto: m6ig2ux7qkmldzsm.onion. Každá skrytá služba představuje jedno (vygenerované) jméno, kterému pak může příslušet jeden nebo více portů.Pokud se podíváte do vytvořeného adresáře pro vaši novou službu, měli byste v něm objevit soubor `hostname` s názvem vaší služby. Název je odvozen ze soukromého klíče, který je k dispozici ve stejném adresáři, v souboru `private_key`. Soubor se soukromým klíčem doporučuji zazálohovat (pokud nechcete generovat nový název po případné ztrátě dat) a chránit proti přečtení nepovolanou osobou (má-li někdo váš privátní klíč, může se vydávat za váš server). Na serveru s Torem je třeba zajistit, aby serverový čas alespoň přibližně odpovídal. Pokud budou systémové hodiny hodně mimo, Tor nemusí být schopný vytvořit okruh.

### Bezpečnost serveru

Co se týče bezpečnosti serveru, na kterém budete provozovat skrytou službu, platí všechny klasické poučky a pravidla. Skryté služby zabezpečte tak, jako by byly vystaveny internetu. Vedle toho doporučuji zvážit, jaké služby budete lidem nabízet. Budete-li uživatelům umožňovat ukládat na server nějaké materiály (od diskusních příspěvků přes články až po obrázky a všemožné soubory), je vysoce pravděpodobné, že se na vašem serveru časem objeví nějaký nelegální materiál. I když toto samozřejmě platí i v případě serverů dostupných z internetu, řekl bych, že tady je to malinko pravděpodobnější. Kromě pravidelné aktualizace a sledování bezpečnostních hlášení vaší distribuce doporučuji navíc sledovat dění kolem Toru jako takového.

### Zachování anonymity

Zachování anonymity je mnohem větší problém. Vhodné nastavení služby, včetně omezení na lokální smyčku (`127.0.0.1`), nemusí stačit (a nejspíše nestačí). Některý serverový software může sdělovat IP adresu serveru nebo poskytovat jiné prozrazující informace, a to buď na regulerní, nebo na podvržené požadavky. Stejně tak, pokud budete provozovat nějakou webovou aplikaci, jakýkoliv úspěšný útok na ni, který povede ke spuštění útočníkova kódu, může IP adresu serveru odhalit. Uživatele (nebo aplikaci běžící pod určitým uživatelem) je možné izolovat z hlediska sítě tak, aby mohl přistupovat pouze k Toru a nikam jinam, a sice pomocí modulu Netfilteru `owner`, který umožňuje vytvářet filtrovací pravidla vztažená na uživatele, takto:

```
iptables -I OUTPUT -m owner --uid-owner uzivatel -j REJECT
iptables -I OUTPUT -d 127.0.0.1 -p tcp --dport 9050 -m owner --uid-owner uzivatel -j ACCEPT
```

Výše uvedené příkazy zařadí dvě pravidla do firewallu, první z nich zakáže uživateli uzivatel odesílat pakety kamkoliv, včetně místních služeb. Druhé pravidlo pak povolí uživateli uzivatel přístup k Toru (127.0.0.1:9050). Tyto dva příkazy se hodí pro zařazení do běžícího firewallu (pravidla se díky parametru -I vkládají vždy na začátek řetězu OUTPUT, tudíž druhé pravidlo se stane prvním, jelikož se přidá na začátek). Pokud firewall stavíte, bude se vám hodit tato syntax (pravidla se přidají na konec řetězce OUTPUT):

```
iptables -A OUTPUT -d 127.0.0.1 -p tcp --dport 9050 -m owner --uid-owner uzivatel -j ACCEPT
iptables -A OUTPUT -m owner --uid-owner uzivatel -j REJECT
```

Toto řešení zabrání tomu, aby aplikace běžící pod daným uživatelem hovořily s někým jiným než s Torem (vyřeší se tím např. problémy s DNS leaky). Slabina tohoto řešení ovšem spočívá v tom, že i takto omezený uživatel může stále velmi snadno získat IP adresu libovolného síťového rozhraní, a poslat ji přes Tor útočníkovi. Ideálním řešením pro server by byla izolace poskytované služby (nebo klienta) na samostatném (fyzickém nebo alespoň virtuálním) serveru umístěném v privátní síti (na síťových rozhraních pak nebudou veřejné IP adresy). Mezi ním a dalším serverem, kde poběží Tor, měl být firewall, který serveru poskytujícímu dané služby umožní komunikovat pouze s Torem na portu 9050.

#### *Fyzická bezpečnost*

Je jasné, že výše uvedená opatření nijak neovlivní situaci, kdy se útočník dostane k serveru fyzicky. Ať už používáte Tor jako klienta, nebo jako prostředníka pro poskytování skryté služby, bude na daném stroji uloženo mnoho dat, která souvisí s používáním Toru (v případě serveru data, která nabízíte, nebo data, která si u vás ukládají vaši uživatelé; v případě Tor klienta cache a historie prohlížeče, stažené soubory apod.). Pokud nechcete, aby se k těmto datům někdo dostal, budete se muset zaměřit i na fyzickou bezpečnost (připomínám, že tento seriál se fyzickou bezpečností a souvisejícím diskovým šifrováním zabýval).

### Závěr

Na závěr nemohu nezopakovat dvě podstatné věci z úvodu. V první řadě doporučuji promyslet, co budete přes Tor nabízet, jak, komu a jaké to pro vás bude mít důsledky. Druhým mým doporučením je zařadit do vašich krizových scénářů nejenom možnost, že server jako takový bude kompromitován, ale že bude kompromitována i jeho identita.

## Správa linuxového serveru: Tvorba Tor relaye

Minulé dva díly představily anonymizační nástroj Tor a jeho možné nasazení při tvorbě anonymizovaného serveru. Dnes vám ukážu, jak vytvořit vlastní Tor relay (resp. Tor server).

### Úvod

Pokud jste nečetli poslední dva díly, přečtěte si přinejmenším první díl miniseriálu o Toru, jelikož obsahuje základní informace o Toru jako takovém, a také klíčové informace o tom, co znamená provozovat Tor relay a jaká jsou případná nebezpečí.

Tor relay, nebo také Tor server, představuje uzel v Tor síti. Tyto uzly jsou používány klienty, kteří z nich vytvoří tzv. okruh, přes který komunikují. Tento okruh se tvoří náhodně a pravidelně se mění. Čím více má Tor síť uzlů, tím lépe je na tom s anonymitou.

### *Druhy Tor serverů*

Existují tři druhy Tor serverů – prostředník (middle relay), výstupní uzel (exit relay) a most (bridge). Prostředník doručuje zašifrované pakety z jednoho Tor uzlu na jiný. Výstupní uzel obsluhuje požadavky „za" klienty, tedy dešifruje příslušný paket, pošle jej do internetu a odpověď pak pošle okruhem klientovi zpět. Tor servery se po svém spuštění zařadí do Tor „adresáře", který představuje seznam dostupných Tor serverů. Tento seznam je veřejný, což umožňuje odpůrcům Toru, aby zakazovali komunikaci klientů s těmito servery. Most (bridge) představuje Tor server, který do veřejného adresáře zařazen není. Mosty tedy mohou být použity těmi, jejichž ISP nebo vláda použití Toru blokuje, aby toto blokování mohli obejít.

### *Nasadit Tor server, nebo ne?*

Vytvoření Tor serveru je třeba pečlivě zvážit, neboť může mít za určitých podmínek pro jeho provozovatele negativní následky. IP adresa serveru se může objevit v některých blacklistech, je-li Tor server zařazen ve veřejném „adresáři". V závislosti na tom, jakou výstupní politiku (exit policy) uplatníte, se může váš server stát i zdrojem podezřelé (nebo přímo nelegální) komunikace (platí pro výstupní uzel, který vyřizuje požadavky „za" klienty). Může se tedy stát, že se stane váš server, nebo přímo vy, objektem policejního vyšetřování nebo případných žalob apod. Viz třeba tento článek. V souvislosti s právními otázkami si můžete také projít právní FAQ o Toru od EFF.

Před tím, než Tor server nasadíte, určitě o tom doporučuji informovat v první řadě správce sítě (nebo ISP), popřípadě správce serveru či hypervizoru. Informovat byste měli všechny, kteří mohou být při případném maléru nějakým způsobem postiženi (zabavení serveru policií apod.). Uvažte také, na jakém serveru budete Tor provozovat, je-li jen váš, nebo zda jej s někým sdílíte (virtuální servery). Rozhodně by to neměl být jakýkoliv produkční či korporátní server. Ideálně by to měl být váš (pokud možno fyzický) server.

Doporučuje se také na IP adrese Tor serveru zřídit webový server a vystavit na něm stránku, která informuje jeho návštěvníky, že se jedná o Tor server. Obsah takové stránky můžete čerpat odsud.

### Instalace

Instalace představuje nainstalování příslušného balíčku s Torem, což v Debianu provedete takto:

```
aptitude install tor
```

### Konfigurace

Nastavení Toru je klíčové. Budete-li tvořit Tor server, rozhodne právě jeho konfigurace o tom, jaký typ (viz výše a níže) postavíte. Konfigurační soubor byste měli najít v /etc/tor/torrc. Dostupných voleb má Tor značné množství, zde budou probrány jen ty nejpodstatnější (o ostatních volbách se dozvíte z manuálové stránky). Základem pro funkci serveru je volba ORPort, která určuje port, který bude ohlášen klientům nebo jiným Tor serverům:

```
ORPort 9001
```

Název může mít 1 až 19 znaků a musí být složen pouze z čísel a písmen. Pokud má váš server více IP adres, můžete definovat, na které IP adrese bude Tor poslouchat:

```
ORListenAddress 1.2.3.4
```

Je také třeba definovat jednoznačný název serveru v parametru Nickname (pozor, nejedná se o DNS jméno, ale nějaký název, který si vy vymyslíte):

```
Nickname mujtorserver
```

Není nutné, ale je doporučeno vytvořit i kontaktní informaci, aby vás mohli správci Toru (nebo kdokoliv jiný) kontaktovat. Mějte na paměti, že tuto e-mailovou adresu mohou z adresáře Tor serverů vyzobat roboti spammerů:

```
ContactInfo Jan Novák <novak AT example dot org>
```

Je také doporučeno, abyste kromě funkce Tor uzlu zprovoznili ještě zrcadlo adresářového serveru, který předává klientům informace o Tor serverech. Toho docílíte definováním portu pro distribuci adresářových informací:

```
DirPort 9030
```

Chcete-li, aby Tor na vašem serveru běžel pouze jako server a nikoliv ještě navíc v režimu klienta, nastavte klientský port na nulu:

```
SocksPort 0
```

Nemáte-li k dispozici neomezený traffic, můžete využít volby pro řízení datového toku. Specifikujte průměrnou hodnotu datového toku a hodnotu náhlé potřeby (burst):

```
BandwidthRate  100KB
BandwidthBurst  1MB
```

Hodnoty jsou v bytech (nikoliv v bitech). Minimum pro průměrnou hodnotu (BandwidthRate) je 20 KB, výchozí hodnota je 5 MB. Můžete také specifikovat strop, tedy datový limit, po kterém Tor server přestane akceptovat nová spojení a vytváření nových okruhů. V tomto případě je třeba definovat hodnotu limitu (v bytech) a údobí, za které se doba počítá, např:

```
AccountingMax 10 GB
AccountingStart week 2 13:00
```

Výše uvedený příklad bude za datový limit považovat 10 GB, přičemž počítat začne týdně (week), vždy v úterý (číslo 2), a to ve 13:00.

Tolik k obecným volbám. Následují specifické volby nutné pro konkrétní typ Tor serveru, který se rozhodnete provozovat.

### *Vytvoření prostředníka (middle relay)*

Prostředník je de facto uzel, který neakceptuje odchozí požadavky. Chcete-li vytvořit prostředníka, postačí nastavit odchozí politiku (exit policy) takto:

```
ExitPolicy reject *:*
```

Tato odchozí politika bude odmítat veškeré pokusy o odchozí spojení.

### *Vytvoření výstupního uzlu (exit relay)*

Zatímco prostředník odchozí komunikaci zcela zakazuje, výstupní uzel by ji naopak měl povolit. Je samozřejmě možné specifikovat, na jaké porty, popřípadě na jaké IP adresy je možné přistupovat, a kam naopak není. Tuto odchozí politiku je možné specifikovat direktivou ExitPolicy. Ta může obsahovat jedno nebo více pravidel oddělených čárkou (pravidla začínají klíčovým slovem accept nebo reject, za ním pak následuje specifikace IP adres a portů). Pravidla se zpracovávají od prvního k poslednímu. Uvažte následující dva příklady:

```
ExitPolicy accept *:80,accept *:443,reject *:*
ExitPolicy reject *:25,reject *:465,accept *:*
```

První příklad definuje velmi restriktivní politiku, která povoluje pouze HTTP a HTTPS (porty 80 a 443), jinou komunikaci nepovoluje. Druhý příklad ukazuje relevantně benevolentní politiku, kde je povoleno vše kromě portů 25 a 465, které odpovídají protokolu SMTP a jeho SSL variantě. Privátní rozsahy a veřejnou IP adresu vašeho serveru není potřeba explicitně blokovat, jelikož jsou blokovány zcela automaticky. Pokud se vám takové chování nezamlouvá, nastavte volbuExitPolicyRejectPrivate na hodnotu 0 a specifikujte odchozí politiku ručně, podle svých představ.

### *Vytvoření mostu (bridge)*

Most je charakteristický tím, že na rozdíl od ostatních typů Tor serverů není automaticky zařazen do veřejného seznamu. Pokud by vás zajímal seznam jako takový, můžete se na něj podívat ve webové podobě třeba sem. Nechcete-li, aby byl váš server veřejně zmiňován, vložte do konfiguračního souboru Toru následující řádek:

```
BridgeRelay 1
```

Vytváříte-li most, nezapomeňte upravit také výstupní politiku (viz vytvoření výstupního uzlu a prostředníka výše).

I když váš server nebude zmiňován veřejně, informace o něm bude stále odesílána, konkrétně pak na seznam mostů. Ten je, jako celek, neveřejný. Zájemci však mohou získat adresy některých mostů třebazde.

***Vytvoření soukromého, tajného serveru***

Pokud nechcete, aby se informace o vašem serveru dostala na jakýkoliv seznam, včetně seznamu mostů, můžete publikování zakázat následující řádkou:

```
PublishServerDescriptor 0
```

Aby měl takový server smysl, měl by jej samozřejmě někdo využívat. Je tedy třeba nějakým způsobem dopravit informaci o jeho existenci (tj. alespoň IP adresu) jeho uživatelům.

**Kontrola funkčnosti serveru**

Po úpravě konfiguračního souboru Tor restartujte:

```
/etc/init.d/tor restart
```

Pokud se vám podařilo Tor server korektně nastavit a vámi nastavené porty Toru jsou zvenčí přístupné, měli byste během 20 minut objevit v logu Toru následující hlášku:

```
Self-testing indicates your ORPort is reachable from the outside. Excellent.
```

Pokud ji neobjevíte, zkontrolujte firewall a konektivitu.

**Bezpečnost Tor serveru**

Abyste zajistili anonymitu klientům sítě Tor, projděte si následující body (další body):

- server pečlivě zabezpečte (bezpečnost byla v tomto seriálu podrobně probírána)
- serverový software i Tor pravidelně aktualizujte
- šifrujte swap a pokud možno šifrujte celý systém (jak na to v Linuxu viz dřívější díly seriálu)
- zálohujte a chraňte obsah /var/lib/tor/keys – kompromitace souborů s klíči umožní útočníkům dešifrovat komunikaci procházející vaším serverem
- pokud neprovozujete webový server, nastavte ORPort na 443 a DirPort na 80
- minimalizujte logování na serveru (ideálně zrušte logy úplně)
- zvýšit bezpečnost můžete provozem Toru na virtuálním serveru nebo v chrootu

***Tor a firewall***

Nastavení odchozí politiky a síťový provoz Toru můžete upravovat a omezovat pomocí linuxového firewallu. Jelikož Tor běží pod konkrétním uživatelem, můžete jej omezit pomocí modulu owner, což bylo naznačeno i s příkladem v minulém díle.

**Závěr**

Na závěr znovu připomínám, co jsem již zmínil v úvodu. Provoz Tor serveru může mít v závislosti na jeho nastavení jisté negativní následky, se kterými byste měli počítat a být na ně připraveni. Je smutným faktem, že Tor jako takový je využíván nejenom etickými způsoby (např. k překonání cenzury), ale také k nejrůznějším neetickým a nezákonným účelům. To je ale z podstaty věci neřešitelný problém. Faktem zůstává, že pro řadu lidí, a to nejen v zemích s vládní cenzurou (Čína, Írán apod.), má Tor obrovský význam. A čím více (důvěryhodných) Tor serverů bude, tím silnější anonymitu a lepší službu bude Tor svým uživatelům poskytovat.

## Správa linuxového serveru: Nastavení sítě pomocí nástroje ip

V tomto dílu vám představím nástroj ip, který již dlouho nabízí alternativu k zastaralému, avšak stále rozšířenému a používanému nástroji ifconfig.

### Úvod a ifconfig

V knihách, článcích, návodech a dokumentaci se často setkáte s nástrojem ifconfig (a dále nástroji arp aroute, které jeho funkčnost doplňují). Možná jste si jej díky tomu osvojili a aktivně jej používáte. Tyto nástroje jsou však již dlouhou dobu považovány za zastaralé, jelikož nerespektují změny v síťovém subsystému linuxového jádra, a to už od řady 2.2. Jejich použití může být kvůli tomu za určitých okolností problematické až potenciálně nebezpečné, pokud si nejste případných úskalí dobře vědomi.

Jistě tušíte, jak nahodit síťové rozhraní a IP adresu. Pomocí nástroje ifconfig byste to provedli takto:

```
ifconfig eth0 down # shození rozhraní eth0
ifconfig eth0 up # nahození rozhraní eth0
ifconfig eth0 192.168.12.12 netmask 255.255.255.0 # nastavení IP adresy a masky
```

Co když ale chcete přiřadit jednomu rozhraní více IP adres? Na serveru je to celkem častý požadavek (např. více IP adres pro SSL virtuální weby). Linux to samozřejmě umí, avšak ifconfig nikoliv. V jeho případě si musíte pomoci jistou obezličkou:

```
ifconfig eth0 192.168.12.12 netmask 255.255.255.0
ifconfig eth0:0 192.168.13.13 netmask 255.255.255.0
ifconfig eth0:1 192.168.14.14 netmask 255.255.255.0
```

V tomto případě je třeba zdůraznit, že rozhraní eth0:0 a eth0:1 jsou sice někde označovaná jako „virtuální", ale ve skutečnosti se jedná o aliasy daných IP adres, tedy aliasy původního rozhraní.

V případě nástroje ip je možné přidávat IP adresy bez jakýchkoliv obezliček:

```
ip addr add 192.168.12.12/24 brd + dev eth0
ip addr add 192.168.13.13/24 brd + dev eth0
```

Problém nastane tehdy, pokud vy (nebo nějaký skript) použije některé z pokročilejších (nebo se starými nástroji nekompatibilních) funkcí nástroje ip. ifconfig vám pak může lhát nebo nasekat neplechu, když přepíše vaše nastavení. Příkladem může být i výše uvedené nastavení více IP adres jednomu rozhraní pomocí nástroje ip. Jelikož nástroj ip běžně k IP adresám aliasy nedefinuje, pak, pokud pomocí něj přiřadíte nějakému rozhraní více IP adres, ifconfig vám je u daného rozhraní nezobrazí:

```
root@debian ~# ip addr add 192.168.12.12/24 brd + dev eth1
root@debian ~# ip addr add 192.168.13.13/24 brd + dev eth1
root@debian ~# ifconfig eth1

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  metric 1
inet 192.168.12.12  netmask 255.255.255.0  broadcast 192.168.12.255
inet6 fe80::20a:aff:febc:1c20  prefixlen 64  scopeid 0x20<link>
ether 00:0a:0a:bc:1c:20  txqueuelen 1000  (Ethernet)
RX packets 629  bytes 42106 (41.1 KiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 11  bytes 678 (678.0 B)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@debian ~# ip addr show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
link/ether 00:0a:0a:bc:1c:20 brd ff:ff:ff:ff:ff:ff
inet 192.168.12.12/24 brd 192.168.12.255 scope global eth1
inet 192.168.13.13/24 brd 192.168.13.255 scope global eth1
inet6 fe80::20a:aff:febc:1c20/64 scope link
valid_lft forever preferred_lft forever
```

Jak je vidět (IP adresy ve výpisech jsou zvýrazněny tučně), ifconfig v tomto příkladu zobrazuje pouze první nastavenou IP adresu, druhou úspěšně ignoruje. Nástroj ip zobrazuje obě.

V zásadě, pokud budete používat výhradně starší nástroje ifconfig, arp a route, a zároveň budete využívat pouze tu funkcionalitu, kterou tyto nástroje umí, neměli byste narazit na problém. Pokud však budete používat nástroj ip, načež si usmyslíte, že použijete ifconfig, můžete v určitých situacích narazit na problém.

Nástroj ip umí nejenom to, co umí nástroje ifconfig, arp a route dohromady, ale také řadu dalších věcí. Kromě toho nástroj ip respektuje současný návrh síťového subsystému jádra Linuxu a jeho možnosti.

### Instalace

Nástroj ip bývá součástí moderních distribucí. Pokud ho ve své instalaci nenajdete, pak vám stačí nainstalovat balíček iproute2, což v Debianu provedete takto:

```
aptitude install iproute2
```

Dlužno dodat, že ve standardní instalaci Debianu Squeeze je již tento balíček obsažen.

### Úvod do nástroje ip

Syntaxe nástroje ip je poměrně odlišná od nástroje ifconfig (a jeho dvou bratrů), avšak umí toho hodně a podporuje krátké zápisy, které vám šetří ruce, pokud je zadáváte ručně.

Prvním parametrem příkazu ip je vždy „modul", se kterým chcete pracovat. To může být síťové rozhraní, IP adresa, směrovací tabulka, ARP tabulka apod.:

```
ip link       # operace se síťovými rozhraními
ip addr       # operace s IP adresami
ip route      # operace se směrovací tabulkou
ip neigh      # operace s ARP tabulkou
ip rule       # operace se směrovacími pravidly
```

Těchto „modulů" je o několik více, nicméně tyto je možné považovat za nejběžnější. Jednotlivé parametry mají samozřejmě svoje zkrácené varianty:

```
ip li         # operace se síťovými rozhraními
ip a          # operace s IP adresami
ip r          # operace se směrovací tabulkou
ip n          # operace s ARP tabulkou
ip ru         # operace se směrovacími pravidly
```

Každý z těchto modulů má svou specifickou syntax. V rychlosti si ji můžete nechat vypsat, pokud budete tápat:

```
ip help
ip addr help
```

Pokud vám nápověda nestačí, můžete použít manuálové stránky:

```
man ip
```

Pro rychlý výpis např. IP adres nemusíte zadávat kromě modulu žádné další parametry:

```
root@debian ~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
```

link/ether 64:31:50:74:1e:c6 brd ff:ff:ff:ff:ff:ff
inet6 fe80::6631:50ff:fe74:1ec6/64 scope link
valid_lft forever preferred_lft forever

**Práce se síťovými rozhraními**

Základní operace se síťovými rozhraními jsou určitě jejich nahození a shození:

```
ip link set dev eth0 up      # nahození rozhraní
ip link set dev eth0 down    # shození rozhraní
```

To je trošku nešťastný začátek, jelikož zrovna v tomto případě je syntaxe nástroje ifconfig úspornější:

```
ifconfig eth0 up
ifconfig eth0 down
```

Při této práci s rozhraními si dejte pozor, abyste nebyli na server připojeni přes SSH, protože pokud shodíte rozhraní, přes které jste připojeni, pak ho už nenahodíte.

Ale zpět k nástroji ip. Ten umí měnit nejenom stav rozhraní, ale také řadu jejich parametrů. Mnoho z nich se dočtete v manuálu, já zmíním dva nejběžnější. Prvním je MAC adresa, kterou můžete změnit i bez nástroje ethtool, takto:

```
ip link set dev eth0 address 11:22:33:44:55:66
```

Dalším parametrem, který můžete chtít někdy změnit, je MTU, tedy maximální velikost IP datagramu, kterou dané rozhraní zpracuje. V případě ethernetu je to nejčastěji 1500 bytů:

```
ip link set dev eth0 mtu 1500
```

Jednotlivá rozhraní si samozřejmě můžete nechat vypsat:

```
root@debian ~# ip link show
1: lo:  mtu 16436 qdisc noqueue state UNKNOWN
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0:  mtu 1400 qdisc pfifo_fast state UP qlen 1000
link/ether 64:31:50:74:1f:c7 brd ff:ff:ff:ff:ff:ff
3: eth1:  mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
link/ether 00:0a:0a:bc:1c:20 brd ff:ff:ff:ff:ff:ff
4: wlan0:  mtu 1500 qdisc mq state UP qlen 1000
link/ether e0:2a:82:45:f4:56 brd ff:ff:ff:ff:ff:ff
```

Můžete si nechat vypsat konkrétní rozhraní:

```
root@debian ~# ip link show dev eth0
2: eth0:  mtu 1400 qdisc pfifo_fast state UP qlen 1000
link/ether 64:31:50:74:1f:c7 brd ff:ff:ff:ff:ff:ff
```

Nebo si můžete nechat vypsat pouze aktivní rozhraní:

```
ip link show up
```

**Práce s IP adresami**

Jak víte, na síti je každý počítač identifikován IP adresou. Abyste tedy mohli komunikovat po síti, musíte nastavit nějakému dostupnému síťovému rozhraní IP adresu. Přidání IP adresy konkrétnímu rozhraní se tu již objevilo:

```
ip addr add 10.0.4.16/24 dev eth1
```

Tato syntaxe přidá danou IP adresu s maskou v CIDR notaci (tzn. /24 je ekvivalentem masky255.255.255.0). Není potřeba nějaký zvláštní parametr pro přidání další IP adresy. Jednomu rozhraní můžete přiřadit tolik IP adres, kolik chcete. Na serverech bývá obvyklé mít více IP adres, ať již IPv4, nebo IPv6.

Připomínám, že můžete použít kratší zápis:

```
ip a a 10.0.4.16/24 dev eth1
```

Chcete-li specifikovat konkrétní broadcast adresu, můžete použít tuto syntaxi:

```
ip addr add 10.0.4.16/24 brd 10.0.4.255 dev eth1
```

Adresu pro broadcast můžete specifikovat v kratší podobě, nejspíše pomocí +, kde se jako broadcast nastaví adresa s jedničkami v těch bitech, které odpovídají síťovému rozhraní v dané síťové masce:

```
ip addr add 10.0.4.16/24 brd + dev eth1
```

IP adresu samozřejmě můžete i zrušit:

```
ip addr del 10.0.4.16/24 dev eth1
```

Máte-li přiřazeno více IP adres jednomu rozhraní a chcete-li je smazat všechny, můžete použít parametrflush:

```
ip a flush dev eth1
```

***Příklad: vytvoření jednoduché místní sítě mezi dvěma počítači***

Pro ilustraci, pokud byste měli dva počítače, propojili je kříženým síťovým kabelem a chtěli si vytvořit jednoduchou síť, postupovali byste tak, že byste nejprve nahodili příslušná rozhraní (obvykle eth0), a poté byste jednotlivým rozhraním na každém počítači přiřadili IP adresu, samozřejmě u každého počítače jinou, ale se stejnou síťovou maskou. Tudíž byste na jednom počítači zadali tyto dva příkazy:

```
ip link set dev eth0 up
ip addr add 10.0.1.1/24 dev eth0
```

A na druhém pak tyto:

```
ip link set dev eth0 up
ip addr add 10.0.1.2/24 dev eth0
```

Poté byste měli být schopni komunikovat:

```
ping 10.0.1.1
```

Tím bych tento díl ukončil. Příště se dozvíte více o práci s ARP tabulkami a o směrování.

## Správa linuxového serveru: Pokročilé nastavení sítě pomocí nástroje ip

Předchozí díl probral základy práce s nástrojem ip. Tento díl osvětlí práci s ARP tabulkou, směrování a nastavení resolveru, tedy zbytek toho, co potřebujete k úspěšnému základnímu nastavení sítě.

### Úvod

Na úvod připomenu minulý díl, který se zabýval základy použití nástroje ip a také trochu jeho porovnáním se stále používaným nástrojem ifconfig, který je však dnes již považován za zastaralý a může působit jisté problémy (více viz minulý díl).

Také bych rád předeslal, že celý tento popis konfigurace předpokládá absenci DHCP serveru na síti. Ten vám obvykle jak přiřadí IP adresu, tak sdělí vašemu počítači, kde najde výchozí bránu i DNS servery. V oblasti nastavení serverů není DHCP příliš obvyklé (resp. ještě jsem se s ním nesetkal), a (nejen) proto je vhodné vědět, jak vše potřebné nastavit ručně.

### Práce s ARP tabulkou

ARP protokol je určen k překladu síťové (IP) adresy na linkovou (MAC) adresu. Pokaždé, když s někým komunikujete po místní síti, se váš počítač nejprve dotáže pomocí ARP protokolu na to, které linkové adrese přísluší daná IP adresa, a té pak odešle příslušný IP datagram. Řada typů útoků v místní síti (zejména pak MITM:man-in-the-middle) jsou vedeny podvržením ARP záznamů.

Vypsat si ARP tabulku je jednoduché:

```
ip neigh show
```

Nebo zkráceně:

```
ip n
```

Do ARP tabulky můžete přidávat záznamy:

```
ip neigh add 192.168.5.50 dev eth0 lladdr 11:22:33:44:55:66
```

Záznamy můžete samozřejmě odebírat:

```
ip neigh del 192.168.5.50 dev eth0
```

Můžete také úplně všechny záznamy vymazat:

```
ip neigh flush dev eth0
```

Přidáním statických záznamů do ARP tabulky se můžete bránit místním MITM útokům. Zde lze také doporučit příslušné nástroje typu *arpwatch* k monitorování ARP provozu na síti a detekování podezřelých aktivit, popřípadě je možné použít nástroj arptables k vytvoření „ARP firewallu", tedy sady filtrovacích pravidel pro protokol ARP. Přidávání statických záznamů do ARP tabulky narazí na problém tehdy, když se MAC adresa daného počítače změní (např. výměnou síťové karty nebo výměnou počítače).

### Nastavení směrování

Základní operací je nepochybně vypsání momentálního obsahu směrovací tabulky:

```
ip route show
```

Nebo zkráceně:

```
ip r
```

Směrovací tabulka se naplňuje určitými typy záznamů automaticky. Kupříkladu, pokud byste nastavili na nějakém rozhraní IP adresu a masku:

```
ip addr add 10.0.5.1/24 dev eth1
```

Pak byste ve směrovací tabulce zjistili záznam odpovídající této nové síti:

```
10.0.5.0/24 dev eth1  proto kernel  scope link  src 10.0.5.1
```

Zásadní informací pro práci v síti (zejména pak v internetu) je mít k dispozici nějaký počítač, kterému můžete poslat všechny pakety, které nenáleží žádné vám známé síti. Takový počítač (přesněji směrovač – router) se obvykle označuje jako brána (gateway). Nastavení brány, konkrétně pak výchozí brány (default gateway), můžete provést takto:

```
ip route add default via 10.0.5.254 dev eth0
```

Rozhraní je zde nepovinné, takže ho můžete vypustit:

```
ip route add default via 10.0.5.254
```

Směrovací tabulku však můžete zaplnit dalšími záznamy. Máte-li bran více (např. pokud jste současně připojeni ke svému ISP a současně ještě k VPN), můžete směrovat určitou komunikaci přes konkrétní bránu. Nebo můžete směrovat určitou komunikaci z jedné sítě do jiné a učinit ze svého linuxového systému směrovač. Ale o tom až za chvíli.

### Nastavení resolveru

Pro plně funkční připojení k internetu vám v tuto chvíli chybí ještě jedna věc. Přiřazení IP adres vašim rozhraním bylo probráno v minulém díle, nastavení výchozí brány bylo probráno kousek výše, zbývá tedy už jen nastavení DNS resolveru. Bez toho byste pro kontaktování jiných počítačů museli místo jmen (např.linuxexpres.cz) použít IP adresy. Resolver je DNS klient, vestavěný v operačních systémech, který se dotazuje resolvujících DNS serverů, jež vám obvykle přiřadí ISP. Ty mají tu charakteristiku, že pro vás realizují vyhledávání, pokud příslušné DNS záznamy nenajdou ve své cache nebo místní databázi.

Nastavení resolveru v Linuxu je velmi jednoduché, stačí upravit soubor /etc/resolv.conf a přidat do něj alespoň jeden záznam pro DNS server:

```
nameserver 1.2.3.4
```

Obvykle vám ISP nabídne alespoň dva, aby vás případný výpadek jednoho nepřipravil o možnost překládat názvy na IP adresy.

### Funkce směrovače

Libovolný linuxový systém může zastat funkci směrovače. To je také jedním z důvodů, proč mnoho „krabiček", kterým se říká směrovače, v sobě Linux obsahuje. Tato funkčnost je však z celkem rozumných bezpečnostních důvodů ve výchozím stavu vypnutá – je tedy třeba ji zapnout. To lze učinit úpravou patřičného parametru v /proc:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Výše uvedená syntaxe se často používá v různých návodech, avšak doporučuje se spíše tato, oficiální varianta:

```
sysctl -w net.ipv4.ip_forward=1
```

Pokud byste chtěli, aby měl váš linuxový systém funkci směrovače po každém startu a ne jen do té doby, dokud nerestartujete, přidejte do souboru /etc/sysctl.conf následující řádek:

```
net.ipv4.ip_forward=1
```

Průchozí komunikaci můžete filtrovat pomocí linuxového firewallu, a to v dedikovaném řetězu FORWARD. Nastavení firewallu se zde věnovat nebudu, neboť toto téma zde již bylo probráno.

### Směrovací tabulka

Nástroj IP umožňuje nejen nastavení výchozí brány, ale i přidávání záznamů do směrovací tabulky, které poslouží k přidání statických cest. To se může hodit, pokud máte takovou konfiguraci sítě, ve které na sebe počítače přímo „nevidí", např. tři počítače propojené (kříženým) síťovým kabelem:

```
počítač 1 ---- směrovač ---- počítač 2
```

Takové nastavení sice není příliš vhodné (je podstatně lepší použít switch nebo bridge), ale ilustruje to úpravu směrovací tabulky. Pro názornost dodám nastavení IP adres:

```
(10.0.0.1) ---- (10.0.0.2) router (10.0.1.1) ---- (10.0.1.2)
```

Aby mohl počítač vlevo komunikovat s počítačem vpravo, musí prostřední počítač propouštět pakety (a fungovat jako router). To je nutná, nikoli však postačující podmínka. Počítač 1 totiž netuší, kde má hledat síť 10.0.1.0/24, a počítač 2 zase netuší, kde má hledat síť 10.0.0.0/24. Proto je oběma počítačům třeba říci, kde se nachází druhá síť a jak se k ní dostat:

```
počítač1 ~# ip route add 10.0.1.0/24 via 10.0.0.2
počítač2 ~# ip route add 10.0.0.0/24 via 10.0.1.1
```

Poté by spolu počítače 1 a 2 měly být schopny komunikovat. Přirozeně, je také možné dané záznamy ze směrovací tabulky vymazat:

```
počítač1 ~# ip route del 10.0.1.0/24 via 10.0.0.2
počítač2 ~# ip route del 10.0.0.0/24 via 10.0.1.1
```

Tolik k jednoduchému statickému routování. Nástroj ip také umožňuje specifikovat pravidla, která mohou provoz směrovat různými způsoby, např. mezi dvě brány (např. ISP a VPN) či do ztracena (ekvivalent akceDROP v linuxovém firewallu). O tom (a nejenom o tom) se dozvíte více v příštím díle.

## Správa linuxového serveru: Další nastavení sítě pomocí nástroje ip

Dnešní díl téma nastavení sítě pomocí nástroje ip uzavře. Představí vám směrovací pravidla, dotkne se protokolu IPv6 a možnosti zřídit si IPv6 tunel přes IPv4 protokol a řekne vám, jakým způsobem změny provedené v minulých dílech nastavit tak, aby přečkaly restart počítače.

Připomínám dva předchozí díly, a sice první díl, který představil nástroj ip a základy práce s ním, zatímcodruhý díl zmínil protokol ARP, základní nastavení směrování a doplnil vše, co nezbytně potřebujete k ručnímu vytvoření připojení k internetu (výchozí bránu, nastavení resolveru a přidání alespoň jedné IP adresy síťovému rozhraní).

### Směrovací pravidla

Nástroj ip (a potažmo linuxové jádro) toho umí mnohem více. Řekněme, že máte k dispozici dvě brány a chcete nějakou komunikaci směrovat přes jednu a jinou komunikaci přes druhou bránu. Něco takového je také možné, a to pomocí směrovacích pravidel (ip rule).

V minulém díle jste viděli příklady směrování. To, co jsem vám zamlčel, je skutečnost, že směrovacích tabulek může být více. Nemusíte tedy mít všechna směrovací pravidla pohromadě, ale můžete je rozdělit. Pomocí ip rule pak můžete definovat, kdy se použije jaká tabulka.

Příkladem může být následující situace. Máte směrovač, který má čtyři síťová rozhraní, přičemž dvě vedou do dvou místních sítí (192.168.0.0/24 a 192.168.1.0/24) a druhá dvě vedou ke dvěma poskytovatelům připojení. Vy chcete pakety z jedné sítě směrovat přes jednoho ISP a pakety z druhé sítě přes druhého ISP. Pro představu doplňuji malý ASCII „obrázek“:

```
síť 192.168.0.0/24 \      / brána 10.0.0.1 --- ISP1
                    router
síť 192.168.1.0/24 /      \ brána 10.0.2.1 --- ISP2
```

Specifikace IP adres na routeru by vypadala takto:

```
ip addr add 10.0.0.254/24 dev eth0
ip addr add 10.0.2.254/24 dev eth1
ip addr add 192.168.0.254/24 dev eth2
ip addr add 192.168.1.254/24 dev eth3
```

Následuje specifikace výchozích bran do dvou oddělených tabulek, tabulky 101 a tabulky 102:

```
ip route add default via 10.0.0.1 dev eth0 table 101
ip route add default via 10.0.2.1 dev eth1 table 102
```

Na závěr je potřeba specifikovat pravidla, která budou posílat pakety do jednotlivých tabulek na základě příslušnosti k určité síti:

```
ip rule add from 192.168.0.0/24 table 101
ip rule add from 192.168.1.0/24 table 102
```

Přirozeně může nastat mnoho jiných situací. Kupříkladu, místo druhého ISP můžete mít VPN (to by se konfigurovalo velmi podobně), nebo místo dvou sítí budete chtít přes jedno připojení směrovat komunikaci, která směřuje na konkrétní počítač, např.:

```
ip rule add to 1.2.3.4 table 101
```

Ve spolupráci s příkladem výše by tento řádek směroval spojení s počítačem 1.2.3.4 přes tabulku 101, tedy ISP1.

Pokud chcete zobrazit obsah konkrétní směrovací tabulky, můžete použít ip route v následující syntaxi:

```
ip route show table 101
ip route show table 102
```

Mohu vám jen doporučit podívat se na kompletní syntax ip rule do manuálových stránek, nebo si alespoň rychle zobrazit nápovědu:

```
ip rule help
```

Zjistíte, že můžete směrovat pakety i podle značek firewallu, dle TOS apod. Zjistíte také, že určité pakety můžete odmítat, a to jak na úrovni směrovacích tabulek (ip route), tak na úrovni směrovacích pravidel (ip route).

### *Maškaráda*

V příkladu se dvěma ISP výše by nastal jeden „drobný“ problém – žádný z počítačů ve vnitřních sítích by se nedostal na internet. Důvodem je použití privátních rozsahů. Počítače ve vnitřní síti totiž nejsou díky nim přímo adresovatelné. V této situaci by bylo třeba zařídit, aby router prováděl SNAT, tedy překlad zdrojové adresy (opakem je DNAT, což je překlad cílové adresy). Konkrétně vám ukážu jeho variantu zvanou maškarádu. Ta spočívá v tom, že jádro nastaví zdrojovou adresu automaticky podle toho, jakou adresu počítač na daném rozhraní má. To má tu výhodu, že pro vytvoření tohoto pravidla nemusíte předem znát IP adresu, kterou bude mít router k dispozici (např. v situaci, kdy ISP přiděluje adresu routeru z DHCP serveru).

Maškaráda (stejně jako SNAT a DNAT) se dají nastavit přes Netfilter, tedy pomocí nástroje iptables:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

Snad nemusím dodávat, že všechno výše uvedené bude fungovat, pouze pokud bude v jádře povoleno předávání paketů (pokud nevíte, jak na to, mrkněte na poslední dva díly, kde je to popsáno). Pokud povoleno nebude, router nebude fungovat jako router a nebude pakety předávat.

### Protokol IPv6 a nástroj ip

Víceméně všechny operace s nástrojem ip, které jste zde viděli, můžete provádět i v rámci IPv6 protokolu. Nástroj ip IPv6 adresy pozná a přizpůsobí se. Někdy se však hodí vědět, jak nástroji ip explicitně říci, že má brát v úvahu konkrétní protokol, buď IPv4 nebo IPv6. K tomuto účelu slouží přepínače -4 a -6:

```
ip -6 addr  # IPv6 adresy všech rozhraní
ip -6 route # IPv6 směrovací tabulka
ip -4 addr  # IPv4 adresy všech rozhraní
ip -4 route # IPv4 směrovací tabulka
```

### IPv6 přes IPv4

V datacentrech byste se neměli setkat s tím, že IPv6 připojení není k dispozici, i když raději onen podmiňovací způsob zdůrazňuji, jelikož do nedávna to ještě stále někde problém byl. Většinou se ale s absencí IPv6 konektivity setkáte spíše v domácích podmínkách než v datacentrech. Ideální je samozřejmě tlačit na ISP, aby vám poskytl nativní IPv6 konektivitu, to v každém případě. Ale do té doby, než se tak stane, můžete využít tunelování IPv6 protokolu přes IPv4 protokol.

K tomu samozřejmě potřebujete někoho, kdo IPv6 konektivitu poskytuje a kdo vám poskytne druhý konec tunelu, vaše IPv6 pakety vybalí z IPv4 paketů a pošle je přes IPv6 síť tam, kam mají odejít. Těchto subjektů je k dispozici celá řada, ideální je samozřejmě vybírat ten subjekt, který je vám fyzicky co nejblíže. Odkážu vás v této souvislosti na seznam na Wikipedii.

Pokud naleznete schopného „brokera“, dá vám k dispozici nejenom potřebné údaje k nastavení sítě, ale i sadu příkazů podle operačního systému a vy můžete použít oblíbenou metodu copy & paste do terminálu či do editoru. Konfigurace pomocí nástroje ip může vypadat nějak takto:

```
ip tunnel add sestkovy-tunel mode sit remote 1.2.3.4 local 4.3.2.1 ttl 255
ip link set sestkovy-tunel up
ip addr add 2001:db8::/32 dev sestkovy-tunel
ip route add ::/0 dev sestkovy-tunel
```

První příkaz vytvoří tunel s názvem sestkovy-tunel v módu sit (což je IPv6-přes-IPv4 tunel). Zde je třeba zmínit dvě IP adresy, a sice adresu remote, tedy adresu koncového bodu tunelu (to je IP adresa vašeho brokera), a adresu local, tedy místní adresu vašeho počítače (tam tunel začíná). Druhý příkaz pak tunel „nahodí“, třetí tunel přidá vámi poskytnutý IPv6 rozsah a následuje nastavení cesty (routy).

### *Zpřístupnění IPv6 konektivity místní síti*

Pokud chcete celé své domácí síti zpřístupnit IPv6 konektivitu, mělo by vám stačit nastavit router avdertisement démona, který bude ohlašovat potřebné údaje počítačům v síti, aby mohli provést autokonfiguraci.

Takovým démonem je například radvd, jehož název může některé mírně zmást. Ne, ono DVD v názvu nemá s dévédéčky nic společného. Je to jen zkrácenina od **r**outer **adv**ertisement **d**aemon.

Ale zpět k radvd. Konfiguruje se v souboru /etc/radvd.conf a příklad konfigurace může vypadat nějak takto:

```
interface eth1 {
AdvSendAdvert on;
prefix 2001:db8::/32 {
AdvOnLink on;
};
```

```
};
```

Je to skutečně velmi jednoduchý příklad, nejprve se specifikuje síťové rozhraní, na kterém se mají oznámení posílat, v tomto případě eth1. Následuje blok s nastavením pro dané rozhraní. První volba v něm zapíná posílání samotných oznámení (tudíž musí být „on"), následuje specifikace prefixu pro danou síť a daný prefix je pak třeba také „povolit" volbou AdvOnLink. Následně démona nastartujte:

```
/etc/init.d/radvd start
```

Pokud se vše povedlo, měly by počítače v místní síti automaticky získat IPv6 adresu i routy.
Více informací o nastavení radvd naleznete v manuálových stránkách:

```
man radvd.conf
```

## Uložení nastavení pro přežití restartu

Pokud budete provádět nastavení sítě pomocí nástroje ip, asi je vám jasné, že tato nastavení nepřečkají restart, jelikož se nikam neukládají. Totéž platí pro nastavení iptables. Aby vaše nastavení přečkalo restart, je třeba ho někam uložit.
Jedním z míst pro ukládání různých procedur a nastavení, jež se mají vykonat po spuštění počítače, je skript /etc/rc.local, který bude spuštěn na konci bootování. Pro nastavení sítě to ale není nejvhodnější umístění, jelikož konec bootování obvykle nastává dávno po aktivaci sítě a síťových rozhraní. Ideální by tedy bylo navázat nastavení sítě na „nahození" příslušného síťového rozhraní. Toho je možné docílit pomocí direktivy up v konfiguračním souboru /etc/network/interfaces:

```
iface eth0 inet static
address 1.2.3.4
netmask 255.255.255.0
gateway 4.3.2.1
up ip addr add 1.2.3.5/24 dev eth0
up ip addr add 1.2.3.6/24 dev eth0
up ip route add 5.6.7.8 via 8.7.6.5
up iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Můžete své nastavení vložit do skriptu a na něj se potom odkázat, nebo všechno napsat do/etc/network/interfaces. Toto řešení je lepší, ale není úplně čisté – ideální by bylo přidat direktivy down, které toto nastavení odčiní, aby se při shození a opětovném nahození rozhraní zbytečně nemnožily záznamy v tabulkách.

## Závěr

Nástroj ip a nastavení sítě by se dalo probírat ještě dlouho, nicméně tímto dílem bych toto téma, alespoň prozatím, ukončil.

## Správa linuxového serveru: Úvod do skenování sítí pomocí Nmap

Nmap je jeden z nejpopulárnějších nástrojů pro skenování portů. Umí toho poměrně dost a výborně se hodí k základním prověrkám bezpečnosti, zejména pak po různých úpravách firewallu nebo konfigurace sítě. V tomto dílu vás s tímto nástrojem seznámím.

### Úvod

Jako správci jste se určitě setkali s potřebou ověřit si, zda vámi nastavený firewall funguje podle vašich představ, zda jste na něco nezapomněli nebo zda jste se v něčem nespletli. Stejně tak jste možná chtěli zjistit, jaké informace mohou o vašich serverech (a službách na nich dostupných) získat případní útočníci.

Nmap je, jak už bylo v úvodu řečeno, jeden z nejpopulárnějších nástrojů pro skenování sítí, alespoň v oblasti GNU/Linuxu. Je to velmi pokročilý nástroj, který má mnoho možností. Umí samozřejmě skenovat porty, ale také zvládne skenovat sítě (a hledat běžící počítače), dokáže detekovat nejenom běžící služby, ale i verze příslušných démonů a také verzi použitého operačního systému.

Na závěr úvodu musím bezpodmínečně upozornit na to, že byste měli skenovat pouze ty počítače nebo sítě, které máte pod svou správou. V ostatních případech byste Nmap měli používat pouze se souhlasem všech dotčených subjektů (majitelé a správci příslušných počítačů nebo sítí). Skenování portů bez svolení může být detekováno a považováno za formu útoku.

Dejte si také pozor na skenování sítí v organizaci, pro kterou pracujete (pracujete-li pro nějakou). Zejména, pokud máte na starosti správu serverů a sítě spravuje někdo jiný. Skenování portů bývá často monitorováno pomocí různých IDS a bylo by nemilé, kdyby se na vás po použití Nmapu vyřítil rozezlený správce sítě.

### Instalace a GUI pro Nmap

Před samotným použitím Nmapu by bylo vhodné jej nainstalovat. Jelikož Nmap je masivně rozšířený v linuxových distribucích, s největší pravděpodobností naleznete stejnojmenný balíček, který můžete nainstalovat pomocí správce balíčků. V Debianu byste to provedli třeba takto:

```
aptitude install nmap
```

V Arch Linuxu a možná i v jiných distribucích je součástí balíčku nmap i jeho GUI, zenmap. V Debianu a Fedoře tomu tak není, tam je balíček s GUI osamostatněn:

```
aptitude install zenmap
```

GUI Nmapu může být poměrně užitečné, jelikož umí značně zpřehlednit výsledky skenování. Umí také kombinovat informace získané z různých skenů, dokonce zvládne i skenované sítě vizualizovat. Určitě tedy doporučuji se na něj podívat, i když v tomto článku se jím zabývat nebudu. Jeho ovládání není nijak komplikované, a pochopíte-li základy práce s Nmapem, měli byste být schopni bez problémů zvládnout práci s jeho GUI.

### Základní použití

Oficiální syntax pro Nmap vypadá takto:

```
nmap [volby] [cíle]
```

V zásadě ale na pořadí nezáleží, Nmap je v tomhle poměrně tolerantní. Na úvod si můžete zkusit oskenovat vlastní počítač:

```
nmap localhost
```

Přirozeně, sken zevnitř a sken zvenčí budou dávat odlišné výsledky, jelikož skenování sebe sama jde přes místní smyčku (local loopback, localhost), kde obvykle běží řada služeb, které ze sítě nemusí být vůbec dostupné. Pokud si vzpomenete na díly o bezpečnosti, zdůrazňoval jsem v nich možnost nakonfigurovat démony tak, aby naslouchaly pouze na místní smyčce a nikde jinde. Např. MySQL je tak (přinejmenším v Debianu) po čerstvé instalaci již nastaven. Dodám, že pro skenování sebe sama nemusí být Nmap vhodným nástrojem, osobně bych doporučil použít netstat:

```
netstat -tulpn
```

Pro úplnost dodám, že volby `-t` a `-u` zobrazují služby běžící na TCP a UDP, dále `-l` je zodpovědná za vypisování pouze naslouchajících služeb (chcete-li zobrazit všechna spojení, použijte `-a`), `-n` vypíná překlad adres na jména (výstup tedy bude vyjádřen v číslech) a konečně `-p` zobrazí PID i název démona, který je zodpovědný za obsazení daného portu. Toto je velmi užitečný nástroj a také velmi užitečná kolona parametrů.

Ale zpět k Nmapu. Většinou jej budete používat na konkrétní počítač na síti:

```
nmap 10.0.1.13
```

Výstup takového příkazu by mohl vypadat takto:

```
Starting Nmap 5.51 ( http://nmap.org ) at 2012-02-26 16:48 CET
Nmap scan report for 10.0.1.13
Host is up (0.019s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.39 seconds
```

Na tomto konkrétním počítači je patrný pouze jediný otevřený port, a sice SSH (port 22). Ostatní porty jsou zavřené.

Pokud nenastavíte žádné volby, využije Nmap pro všechny parametry výchozí hodnoty. Ty zahrnují mj. typ skenu, rozsah portů, rychlost skenování apod. Nmap také volí různé hodnoty v závislosti na tom, jaká máte privilegia. Kupříkladu, pouštíte-li jej pod rootem, provádí TCP SYN scan (`-sS`), zatímco pokud byste jej pustili pod neprivilegovaným uživatelem, kde není k dispozici přímý přístup k síťovému rozhraní, zvolí Nmap náhradní variantu, TCP connect scan (`-sT`). Skutečnost, že různé funkce Nmapu vyžadují různá privilegia, byste měli mít na paměti, jelikož Nmap ne vždy varuje, že mu privilegia chybí. Pak se můžete dlouho divit, proč vrací prázdný výsledek, i když jste parametry specifikovali dobře.

Ve výchozím stavu provádí Nmap před samotným skenováním portů test, zda je příslušný počítač online (host discovery). Pokud v této fázi nezjistí, že je počítač online, nebude provádět skenování portů:

```
nmap 10.0.1.254

Starting Nmap 5.51 ( http://nmap.org ) at 2012-02-28 17:28 CET
Note: Host seems down. If it is really up, but blocking our ping probes, try -PN
Nmap done: 1 IP address (0 hosts up) scanned in 3.00 seconds
```

Zde naštěstí Nmap rovnou radí, že zjištění stavu lze přeskočit pomocí volby `-PN`:

```
nmap -PN 10.0.1.254

Starting Nmap 5.51 ( http://nmap.org ) at 2012-02-28 17:28 CET
Nmap scan report for orion (10.0.1.254)
Host is up (0.00035s latency).
Not shown: 998 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
53/tcp  open   domain

Nmap done: 1 IP address (1 host up) scanned in 4.56 seconds
```

Ke zjištění stavu cílového počítače využívá Nmap různé nástroje – kromě klasického ICMP echo (ping) využívá také protokol ARP, ovšem pouze v privilegovaném režimu (viz varování výše).

### Skenování sítí

Nmap umí skenovat nejenom konkrétní počítače, ale také celé sítě. Realizuje se to velmi snadno, prostě specifikujete konkrétní subnet místo jednoho počítače:

```
nmap 10.0.1.0/24
```

Pokud nechcete provádět skenování portů jednotlivých počítačů, ale pouze zjištění, které počítače jsou v dané síti dostupné, můžete skenování portů vypnout (volbou `-sP`):

```
nmap -sP 10.0.1.0/24

Starting Nmap 5.51 ( http://nmap.org ) at 2012-02-28 16:54 CET
Nmap scan report for 10.0.1.12
```

Host is up (0.00013s latency).
Nmap scan report for 10.0.1.13
Host is up (0.014s latency).
Nmap scan report for 10.0.1.15
Host is up (0.0018s latency).
Nmap scan report for 10.0.1.16
Host is up (0.00059s latency).
Nmap scan report for 10.0.1.17
Host is up (0.0017s latency).
Nmap scan report for 10.0.1.51
Host is up (0.00097s latency).
Nmap scan report for 10.0.1.252
Host is up (0.015s latency).
Nmap done: 256 IP addresses (7 hosts up) scanned in 3.01 seconds

**Interpretace výsledků**

Jednotlivé porty se mohou jevit buď otevřené (open), zavřené (closed), nebo filtrované (filtered). Otevřený port znamená dostupnou službu a potenciální prostor pro průnik útočníka do systému. Služba, která je zvenčí dostupná, může být (a bývá) vystavena nejrůznějším útokům, a měla by tedy být velmi dobře zabezpečena. Pokud nemusí být přístupná světu (např. SSH), bývá dobré ji na firewallu omezit pouze pro ty počítače a sítě, ze kterých se na ni potřebujete hlásit.

Zavřený port je takový, který odmítá pokus o spojení (v iptables je to typicky -j REJECT). Tím se liší od filtrovaného portu, který neodpovídá vůbec (ekvivalent -j DROP). Pokud chcete být hodní a chovat se podle norem, měli byste na pokus o spojení reagovat odmítnutím. To však podstatně urychluje skenování portů. Mimo jiné i proto řada správců využívá možnosti zahazovat neoprávněné žádosti o spojení bez jakéhokoliv oznamování. Skenování portů pak trvá déle, jelikož nelze s jistotou říci, zda paket k cíli dorazil nebo ne, tudíž se déle čeká. Skenování portů ovšem zahazování žádostí o spojení nijak nezabrání.

## Správa linuxového serveru: Typy skenů a volby Nmapu

Minulý díl se věnoval základům použití nástroje Nmap, který je jedním z nejpoužívanějších nástrojů pro skenování portů (a sítí). Tento díl naváže na předchozí a představí jednotlivé typy skenů a volby Nmapu.

Na úvod bych rád zmínil skutečnost, že volby Nmapu se mezi jednotlivými verzemi mohou měnit. To bohužel poznamenalo minulý díl (který doporučuji si přečíst, pokud jste tak ještě neučinili), kde jsem zmiňoval volby -sn a -Pn, které starší Nmap dostupný v Debianu Squeeze nezná. Jejich ekvivalentem jsou volby -sP a -PN, které vám v Debianu Squeeze budou fungovat. Za tento šotek se omlouvám, v dotyčném článku už je to opraveno. V případě problémů s nevyhovující syntaxí doporučuji nahlédnout do manuálu, který bude zmiňovat specifika té verze, kterou na svém stroji používáte.

### Rychlost skenování

Jednou z podstatných voleb je rychlost skenování. Příliš velká rychlost zatíží síť i server, naopak velmi nízká slouží především k obcházení IDS (Intrusion Detection System). Nmap má k dispozici několik šablon dostupných pod volbou -T. Tyto šablony jsou paranoid, sneaky polite, normal (výchozí), aggressive ainsane. Jsou seřazeny od nejpomalejší k nejrychlejší. První dvě jsou určené k obcházení IDS, polite šetří síť i cílový počítač, normal je výchozí volba, aggressive předpokládá, že mezi vámi a cílem je rychlá síť, ainsane obětuje přesnost skenu ve snaze dosáhnout co nejvyšší rychlosti. Pokud máte tedy např. cíl v LAN, můžete použít volbu aggressive:

```
nmap -T aggressive 10.11.12.13

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 14:02 CET
Interesting ports on 10.11.12.13:
Not shown: 997 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds
```

Dodám, že místo těchto šablon je možné parametry skenování nastavovat přímo. Více o tom se dozvíte v sekci *TIMING AND PERFORMANCE* manuálu k Nmapu.

### Specifikace portů ke skenování

Ve výchozím stavu skenuje Nmap celkem 1000 nejpoužívanějších portů. Jak jistě víte, čísla portů se pohybují v rozmezí 0–65535. Pokud tedy někdo např. přesunul SSH port z 22 na 65022, tak se o tom nedozvíte, dokud nenastavíte rozsah ručně.

Nmap nabízí celkem tři varianty specifikace portů. Pokud se vám 1000 nejpoužívanějších portů zdá moc, můžete Nmapu nařídit, aby skenoval pouze 100 nejpoužívanějších portů, a to parametrem -F:

```
nmap -F -v 10.11.12.13

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 14:02 CET
Interesting ports on 10.11.12.13:
Not shown: 97 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

Třetí variantou je ruční specifikace portů, která se provádí volbou -p, takto:

```
nmap 10.11.12.13 -p 22,136-139,440-450

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 14:04 CET
Interesting ports on 10.11.12.13:
PORT    STATE  SERVICE
22/tcp  open   ssh
136/tcp closed profile
137/tcp closed netbios-ns
138/tcp closed netbios-dgm
139/tcp open   netbios-ssn
440/tcp closed sgcp
441/tcp closed decvms-sysmgt
442/tcp closed cvc_hostd
443/tcp closed https
444/tcp closed snpp
445/tcp open   microsoft-ds
446/tcp closed ddm-rdb
447/tcp closed ddm-dfm
448/tcp closed ddm-ssl
449/tcp closed as-servermap
450/tcp closed tserver
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

Na závěr dodám, že Nmap provádí implicitně tzv. randomizaci portů, což znamená, že místo toho, aby skenoval postupně jeden po druhém od prvního k poslednímu, vybírá porty náhodně. Pokud byste z nějakého důvodu toto chování nepreferovali, můžete ho vypnout volbou -r.

### Typy skenů

TCP SYN sken, -sS, je výchozím typem skenu a funguje tak, že posílá pakety s příznakem SYN, tedy pakety, které zahajují tzv. 3-way handshake (vytvoření TCP spojení). Je-li port otevřen, měl by server odpovědět paketem s příznakem SYN/ACK (popř. SYN), což Nmap zaznamená. Jelikož k dokončení spojení je potřeba finální odpovědi ze strany klienta, která při tomto typu skenu samozřejmě posílána není, nedochází tak k vytváření TCP spojení. Proto se tento typ skenu označuje jako „stealth". Jednotlivé běžící služby by tento typ skenu neměly zachytit a zalogovat. Naopak IDS bývají schopné tento typ skenu detekovat.

TCP connect sken, -sT, je výchozím typem skenu, pokud nemáte práva k přímému přístupu k síťovému rozhraní. Zde dochází k vytváření TCP spojení, což se může projevit v logu jednotlivých služeb.

UDP sken, -sU, jak už jeho název naznačuje, skenuje UDP protokol. Lze jej kombinovat s TCP skeny a dosáhnout tak oskenování obou protokolů naráz. UDP sken je trošku sofistikovanější (a také podstatně pomalejší). V případě některých typických UDP portů (jako např. DNS) se posílají pakety s ohledem na daný protokol, jinak se pošle UDP paket a čeká se na odpověď. Pokud odpověď přijde, je port považován za otevřený, jinak získá statut open|filtered – tento poněkud matoucí statut se vám pokouší sdělit, že daný port otevřený být může (jen nedošla odpověď) i nemusí (může být filtrovaný firewallem). Zde může pomoci detekce verzí (-sV, více o detekci verzí viz dále):

```
nmap -sU 10.11.12.13 -p 136-139
```

```
Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 14:34 CET
Interesting ports on 10.11.12.13:
PORT     STATE         SERVICE
136/udp closed        profile
137/udp open|filtered netbios-ns
138/udp open|filtered netbios-dgm
139/udp open|filtered netbios-ssn
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 2.59 seconds

debian:~# nmap -sU 10.11.12.13 -p 136-139 -sV

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 14:34 CET
Interesting ports on 10.11.12.13:
PORT     STATE         SERVICE      VERSION
136/udp closed        profile
137/udp open          netbios-ns   Microsoft Windows XP netbios-ssn
138/udp open|filtered netbios-dgm
139/udp closed        netbios-ssn
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)
Service Info: Host: LOCALHOST; OS: Windows

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 57.58 seconds
```

Jak je z příkladu výše patrné, detekce verzí mírně vylepšila výsledky skenu a nalezla otevřený UDP port 137.
Toto jsou základní typy skenů, přičemž Nmap jich umí pochopitelně více. Tyto pokročilejší až pokročilé skenovací techniky využívají obvykle různé skuliny specifikace protokolu TCP k rozlišení otevřeného portu od zavřeného (např. TCP NULL, FIN a Xmas skeny). Bývají ještě obtížněji detekovatelné než TCP SYN sken, avšak nelze vyloučit detekci vhodně nakonfigurovaným IDS. Stejně tak nebývají jednoduché na interpretaci výsledků a nezaručují použitelné výsledky u některých síťových prvků a operačních systémů. Tyto typy skenů jsou dobře popsány v manuálu Nmapu a vzhledem k tomu, že se jedná o skutečně pokročilé techniky, pouze vás na ně odkážu.

**Detekce verzí služeb**

Nmap je vám schopen zjistit nejenom otevřené porty pomocí nejrůznějších technik (viz výše), ale také to, o jaké služby se jedná a které verze démonů tam běží. Tato vlastnost je nedocenitelná, i když je třeba si uvědomit, že v případě jejího použití se zbavujete výhod „stealth" typů skenu (jelikož detekce verzí je mírně invazivní a jistě bude zaznamenána v logu daných služeb). Detekce verzí služeb funguje tak, že po zjištění všech otevřených portů Nmap pomocí databáze služeb zjišťuje, co na oněch portech běží. Představu si můžete udělat na následujícím příkladu:

```
nmap -sS 10.11.12.13

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 15:02 CET
Interesting ports on 10.11.12.23:
Not shown: 997 closed ports
PORT     STATE SERVICE
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
1025/tcp open  NFS-or-IIS
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds

debian:~# nmap -sS 10.11.12.13 -sV

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 15:02 CET
Interesting ports on 10.11.12.13:
Not shown: 997 closed ports
PORT     STATE SERVICE     VERSION
139/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: MYGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: MYGROUP)
1025/tcp open  ssh         OpenSSH 5.3 (protocol 2.0)
MAC Address: 52:54:00:29:69:03 (QEMU Virtual NIC)

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.14 seconds
```

Jak je vidět, bez použití detekce verzí vám Nmap vrátí typ služby z /etc/services, tedy typ služby, která se obvykle na daném portu provozuje, což nemusí vždy odpovídat. To je demonstrováno v druhém skenu, kde se díky detekci verzí služeb podařilo odhalit, že na portu 1025 běží ve skutečnosti OpenSSH, a to verze 5.3.

**Detekce operačního systému**

Poslední zajímavou funkcí Nmapu, kterou zmíním, je schopnost detekovat verzi použitého operačního systému. Tato funkce nefunguje vždy zcela spolehlivě a ne vždy se jí povede operační systém detekovat, popřípadě detekovat korektně. Pro rozumnou šanci úspěšné detekce je třeba, aby cíl měl alespoň jeden otevřený port.

```
nmap -O -sS 10.11.12.254

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 15:08 CET
Interesting ports on orion (10.11.12.254):
Not shown: 997 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
53/tcp  open   domain
667/tcp closed unknown
MAC Address: 00:0D:B9:17:ED:DD (PC Engines GmbH)
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.15 - 2.6.26, Linux 2.6.18-8.el5 (Red Hat Enterprise Linux 5), Linux 2.6.21 (Arch Linux 0.8, x86)
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.00 seconds
```

V příkladu výše je detekce úspěšná – operační systém je skutečně Linux, konkrétně jádro 2.6.26, což bez chyby odpovídá dané specifikaci. Jednalo se ovšem o zcela ideální podmínky (počítač byl připojen přímo na síti, mezi ním a Nmapem nebyl jediný router).

### Nmap a -A

Poměrně zajímavou volbu tvoří -A, což je „šablona" zapínající mnoho typických funkcí (detekci verzí služeb i operačního systému, traceroute a použití skriptů z kategorie „default"). Skripty jsou psané v jazyce LUA a představují možnosti rozšíření funkcionality Nmapu. Některé přibalené skripty mohou být poměrně invazivní (v kategorii „default" by měly být zejména bezpečné skripty). Skripty umí provádět ledacos, v příkladu níže je u SSH patrné vypsání fingerprintů serverových klíčů. Více o skriptech se dozvíte zde.

Výstup Nmapu s použitím volby -A může vypadat takto:

```
nmap -A 10.11.12.254

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-11 15:16 CET
Interesting ports on orion (10.11.12.254):
Not shown: 997 filtered ports
PORT    STATE  SERVICE VERSION
22/tcp  open   ssh     OpenSSH 5.1p1 Debian 5 (protocol 2.0)
| ssh-hostkey: 1024 38:9f:28:de:40:c0:19:fa:2c:b8:57:c2:c9:61:6e:c8 (DSA)
|_ 2048 00:7d:67:3d:ab:d9:14:e5:af:09:72:91:14:ff:9f:f3 (RSA)
53/tcp  open   domain  ISC BIND unbound 1.4.6
667/tcp closed unknown
MAC Address: 00:0D:B9:17:ED:DD (PC Engines GmbH)
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.15 - 2.6.26, Linux 2.6.18-8.el5 (Red Hat Enterprise Linux 5)
Network Distance: 1 hop
Service Info: OS: Linux

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.05 seconds
```

### Hlavně opatrně

Nmap je velmi schopný síťový skener, který můžete použít nejenom v GNU/Linuxu a unixových operačních systémech, ale také třeba na Windows. Můžete jej používat v příkazové řádce i pomocí grafického rozhraní, které umí např. kompletovat výsledky z více skenů, popřípadě vizualizovat skenovanou síť. Alespoň základní zacházení s Nmapem patří k základům schopností správců serverů i sítí.

Při používání Nmapu je vhodná opatrnost a dobrá představa o tom, čemu vlastně svůj cíl vystavíte. Nmap umožňuje provádět celkem invazivní operace, které mohou za určitých okolností způsobit problémy. Dobrá představa o tom, co Nmap vlastně provádí, je také nutná ke správné interpretaci výsledků. Rozhodně není vhodné spoléhat na to, že to, co Nmap napíše, vždy na sto procent platí. Je třeba vědět, odkud se ta informace vzala a jak byla získána. Kupříkladu, bez detekce verzí se služba na portu identifikuje dle záznamu v /etc/services, což samozřejmě nemusí odpovídat, jak bylo výše demonstrováno. Určitě tedy doporučuji experimentování (ve vlastní síti) a studium dokumentace k Nmapu.

Tím bych problematiku Nmapu ukončil.

## Správa linuxového serveru: Úvod do poštovního serveru

Tímto dílem začíná menší exkurze do světa poštovních serverů a poštovních služeb. Podíváme se na základní terminologii a základní principy elektronické pošty.

### Úvod

Elektronická pošta představuje jednu z nejzákladnějších, nejpoužívanějších a také nejstarších služeb internetu. Z pohledu správce představuje poštovní server jednu z nejnáročnějších služeb na konfiguraci, správu a zabezpečení, což platí dvojnásob, jedná-li se o větší nebo firemní poštovní server.

Pokud se na elektronickou poštu podíváte podrobněji, zjistíte, že se v žádném případě nejedná o jednu jedinou službu. Jedná se o řadu různě provázaných služeb používajících různé mechanismy a různé protokoly. Prakticky na každé úrovni máte k dispozici alternativy a široké možnosti nastavení. Díky tomu se nastavení poštovního serveru stává komplexní, až téměř tvůrčí činností.

Vezměme to ale popořadě. Z čeho všeho se tedy skládá nebo může skládat taková poštovní služba?

### MTA: Mail Transport Agent

Tato komponenta je patrně tou nejzásadnější částí celého mechanismu – MTA je služba zodpovědná za přenos pošty z jednoho počítače na druhý, a to prostřednictvím klient–server architektury (server poštu přijímá, klient posílá). Poštu k odeslání může MTA získat buď z jiného MTA (pak se jedná o tzv. „relay“, tedy předání zprávy jinému „poštovnímu serveru“), nebo od MUA (pod čímž si zatím představte např. Thunderbird).

V GNU/Linuxu je k dispozici řada MTA. Tento seriál se bude zabývat Postfixem, nicméně existuje celá řada dalších jako sendmail, Exim, qmail atd.

#### *SMTP*

K přenosu zpráv je využíván protokol SMTP (port 25). Jedná se o velmi starý protokol, který byl postupem času poměrně dost rozšiřován, nicméně základní principy zůstaly stejné. Jedná se o klasický textový protokol, kde klient zadává určité příkazy a server na ně reaguje. To, jak protokol SMTP vypadá v praxi, můžete vidět v příkladu níže:

```
server: 220 mail.example.org ESMTP Postfix (Debian/GNU)
klient: EHLO client.example.org
server: 250-mail.example.org
server: 250-PIPELINING
server: 250-SIZE 10240000
server: 250-ETRN
server: 250-STARTTLS
server: 250-AUTH LOGIN PLAIN
server: 250-AUTH=LOGIN PLAIN
server: 250-ENHANCEDSTATUSCODES
server: 250-8BITMIME
server: 250 DSN
klient: MAIL FROM: michal.docekal@example.org
server: 250 2.1.0 Ok
klient: RCPT TO: jan.novak@example.com
server: 250 2.1.5 Ok
klient: DATA
server: 354 End data with <CR><LF>.<CR><LF>
klient: From: "Michal Docekal" <michal.docekal@example.org>
klient: To: "Jan Novak" <jan.novak@example.com>
klient: Date: Tue, 20 Mar 2012 10:54:50 +0100
klient: Subject: Predmet zpravy
klient:
klient: Ahoj,
klient:
klient: posilam velmi dulezitou zpravu.
klient:
klient: Michal Docekal
klient:
klient: .
server: 250 2.0.0 Ok: queued as D882C297011E
klient: QUIT
server: 221 2.0.0 Bye
```

Přirozeně, označení „klient“ a „server“ byly do výpisu přidány za účelem objasnění rolí klienta a serveru, nejsou součástí protokolu SMTP.

Jak je patrné, relace doručování e-mailu začíná příkazem EHLO (u hodně starých MTA/MUA to může být iHELO). Zde se klient identifikuje serveru – parametrem EHLO by mělo být plně kvalifikované doménové jméno klienta, přinejhorším specifikace IP adresy (IPv4 adresa se vkládá do hranatých závorek: [1.2.3.4]).

Klient dále pokračuje specifikací odesílatele MAIL FROM:, specifikací příjemce nebo příjemců RCPT TO:, načež následuje příkaz DATA, po kterém následuje tělo zprávy. Tělo začíná hlavičkami jako From:, To:, Date: a samozřejmě nechybí ani předmět zprávy Subject:. Text zprávy je ukončen sekvencí znaků konce řádku, tečky a dalšího konce řádku. Klient ukončuje relaci příkazem QUIT. Všimněte si také reakcí serverů, které jsou zde více či méně samovysvětlující. Máte-li zájem dozvědět se o SMTP protokolu více, podívejte se naRFC 5321, kde je nejnovější revize protokolu SMTP z roku 2008.

Interakci s MTA pomocí protokolu SMTP si můžete vyzkoušet sami, ručně, a to buď pomocí nástroje `telnet`, nebo `nc`:

```
telnet postovni-server.example.org 25
nc postovni-server.example.org 25
```

Tento postup je velmi vhodný také pro testování a diagnostiku vlastních poštovních serverů (a nejen poštovních serverů, tohoto postupu lze využít u jakéhokoliv textového protokolu), přeci jen, možnost si ručně vyzkoušet, jak se váš server chová v různých situacích, je nedocenitelná pomůcka.

#### *DNS*

Funkcionalita SMTP je přímo závislá na DNS. Asi vám nemusím připomínat, že e-mailová adresa má nejčastěji tvar jmeno@domena. Otázkou je, podle čeho určí MTA server, na který má poštu doručit. Odpovědí jsou tzv. MX záznamy příslušné domény. Ty specifikují jeden nebo více poštovních serverů a přiřazují jednotlivým serverům danou prioritu. Priorita je číslo, kde nižší číslo odpovídá vyšší prioritě (podobně jako „niceness“ u unixových procesů). MX záznamy si můžete poměrně snadno vypsat:

```
host -t MX example.org
```

Nebo, pokud preferujete nástroj dig:

```
dig -t MX example.org
```

Pro ilustraci, výstup prvního z uvedených příkazů může vypadat např. takto:

```
example.org mail is handled by 20 mail-backup.example.org.
example.org mail is handled by 10 mail.example.org.
```

Z výpisu je patrné, že MX záznamy domény example.org jsou v tomto případě dva. Nejvyšší prioritu má server mail.example.com, nižší prioritu má pak mail-backup.example.org. To znamená, že pokud se MTA nepodaří doručit poštu serveru s nejvyšší prioritou, měl by zkusit server s nižší prioritou. V případě, že má několik serverů stejnou prioritu, vybere si z nich MTA náhodně.

Pokud pro danou doménu neexistuje vůbec žádný MX záznam, použije se A záznam dané domény místo něj.

#### *Nastavení poštovního serveru versus SMTP*

Poštovní server lze nastavit různým způsobem. Můžete respektovat příslušná RFC, což určitě patří k „dobrým mravům“ správců serverů. Naneštěstí bývá podstatně jednodušší nechat se strhnout úmyslem efektivnějšího boje se spamem (popřípadě neznalostí RFC) a RFC porušit. Kupříkladu, některé servery jako parametr EHLO vyžadují plně kvalifikované doménové jméno, i když podle RFC tam být nemusí. Takových příkladů bychom určitě našli mnoho.

Je třeba mít na paměti, že čím striktněji svůj server nastavíte, tím méně pošty budete dostávat, a to platí pro spam i pro regulérní poštu, kterou dostávat chcete. Bohužel, v tomhle nepomáhá vždy ani důsledné respektování RFC, jelikož správci poštovních serverů je ne vždy správně nastaví (důvodem je mj. relativní složitost poštovních služeb zmíněná v úvodu), což, paradoxně, činí správu poštovních serverů ještě složitější.

***Anatomie e-mailu***

Pokud jste tak ještě nikdy neučinili, bylo by velmi vhodné si vyhlédnout jednu nebo několik zpráv ve vaší poštovní schránce a podívat se na jejich „zdrojový kód". Ten může vypadat např. takto:

```
Return-Path: <michal@example.cz>
X-Spam-Checker-Version: SpamAssassin 3.3.1 (2010-03-16) on example.net
X-Spam-Level:
X-Spam-Status: No, score=-1.9 required=5.0 tests=BAYES_00,SPF_HELO_PASS,
SPF_PASS,T_DKIM_INVALID autolearn=ham version=3.3.1
X-Original-To: michal@example.net
Delivered-To: michal@example.net
Received: from mail.example.cz (mail.example.cz [1.2.3.4])
(using TLSv1 with cipher ADH-AES256-SHA (256/256 bits))
(No client certificate requested)
by example.net (Postfix) with ESMTPS id 84BA8297011E
for <michal@example.net>; Tue, 20 Mar 2012 16:03:33 +0100 (CET)
Received: from localhost (localhost.localdomain [127.0.0.1])
by example.cz (Postfix) with ESMTP id 27788214C37
for <michal@example.net>; Tue, 20 Mar 2012 16:03:33 +0100 (CET)
X-Virus-Scanned: Debian amavisd-new at server.example.cz
Received: from server.example.cz ([127.0.0.1])
by localhost (server.example.cz [127.0.0.1]) (amavisd-new, port 10024)
with ESMTP id G9yD4RqiBxFh for <michal@example.net>;
Tue, 20 Mar 2012 16:03:32 +0100 (CET)
Received: from localhost.localdomain (unknown [4.3.2.1])
(using TLSv1 with cipher DHE-RSA-AES128-SHA (128/128 bits))
(No client certificate requested)
by server.example.cz (Postfix) with ESMTPSA id A759A214C36
for <michal@example.net>; Tue, 20 Mar 2012 16:03:31 +0100 (CET)
Date: Tue, 20 Mar 2012 16:03:25 +0100
From: Michal =?UTF-8?B?RG/EjWVrYWw=?= <michal@example.cz>
To: michal@example.net
Subject: =?UTF-8?B?VGVzdG92YWPDrQ==?= e-mail
Message-ID: <20120320160325.5df8d7b8@example.cz>
X-Mailer: Claws Mail 3.8.0 (GTK+ 2.24.10; x86_64-unknown-linux-gnu)
Mime-Version: 1.0
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

Ahoj,

pos=C3=ADl=C3=A1m testovac=C3=AD e-mail.

Michal Do=C4=8Dekal
```

Všimněte si, že si e-mail s sebou tahá veškerou historii, kterými servery e-mail procházel (hlavičkyReceived:). Tuto historii čtěte v pořadí od posledního záznamu k prvnímu. Všimněte si, že jedním serverem může e-mail procházet více než jednou (e-mail je zpracováván různými dalšími službami a poté opět předán SMTP serveru). Jsou zde patrné i hlášky těchto dalších služeb (amavis, spamassassin atd.).

Samotné tělo obsahuje zprávu v kódování UTF-8, avšak jelikož SMTP protokol je skutečně čistě textovým protokolem, všechny netextové znaky musí být zakódovány (zde je použito „quoted-printable"). Tento postup samozřejmě zvětšuje objem přenášených dat, což je nejvíce patrné v případě přenosu binárních dat (e-mailových příloh), kde dojde v důsledku kódování ke zvýšení objemu přenášených dat přibližně o 1/3.

Jednotlivé parametry hlavičky mailu (např. pole From:) obvykle nejsou kontrolovány (ono v podstatě ani není jak je důsledně zkontrolovat) a mohou obsahovat různé hodnoty (včetně nesmyslných nebo podvržených). Není tedy problém vytvořit si „falešnou identitu". To je problémem stáří a snahy zachování zpětné kompatibility protokolu SMTP – bezpečnost (MITM útoky), soukromí (e-mail je v řadě případů posílán a vždy skladován nešifrovaný, v čisté textové podobě) nebo autentizace (možnost si ověřit, že komunikuji skutečně s tím, s kým komunikuji) nebyly brány v úvahu při jeho tvorbě. Toto do jisté míry řeší nástroje jako GnuPG či PGP, které umožňují e-maily podepisovat a šifrovat. V praxi se, alespoň v mém případě, ukazuje jako největší problém neochota komunikačních protistran podepisovat a šifrovat.

Tím bych tento díl ukončil. Příště se budu věnovat zbytku úvodu do problematiky poštovního serveru.

## Správa linuxového serveru: Agenti e-mailu, metody ochrany proti spamu

Tento díl pokračuje v menší exkurzi do světa elektronické pošty, snaží se dovysvětlit pojmy z této oblasti a popsat problematiku nevyžádané pošty.

### Úvod

Na úvod připomínám minulý díl seriálu, kde byly zmíněny některé základní pojmy, dále charakter e-mailu i SMTP protokolu, a také vazba na službu DNS. Pokud jste jej dosud nečetli, doporučuji vám se na něj podívat.

### MDA: Mail Delivery Agent

V minulém díle zazněl pojem MTA, tedy Mail Transport Agent, program zodpovědný za přenos pošty z jednoho počítače na druhý. MDA je program zodpovědný za doručení e-mailu do poštovní schránky. V GNU/Linuxu je to nejčastěji procmail či maildrop.

Doručování zpráv může být řízeno i přizpůsobováno jednotlivým uživatelům, a to obvykle právě na úrovni MDA. Kupříkladu, *maildrop* má svůj hlavní konfigurační soubor v /etc/maildroprc, ale je také možné poštu filtrovat na úrovni jednotlivých uživatelů, a sice v souboru .mailfilter v jejich domovském adresáři. Můžete tak vytvářet pravidla pro přesměrování nebo přesun mailů do příslušných složek.

K dispozici je dokonce specifický jazyk pro pokročilé filtrování e-mailů s názvem Sieve (více viz odkazy pod článkem). Konfigurace MTA obvykle odkazuje na konkrétní MDA, který má být použit (v Postfixu k tomu slouží konfigurační direktiva mailbox_command, která udává, jaký program se má použít k doručování pošty).

#### *Poštovní schránka*

Aby to nebylo jednoduché, neexistuje jeden způsob uchovávání pošty. Nejčastěji se setkáte s jedním ze dvou způsobů, a sice mboxu nebo maildir. Mbox ukládá všechnu poštu do jednoho velkého souboru, zatímco maildir ukládá poštu do jednotlivých souborů ve struktuře adresářů. Tuto strukturu si můžete vytvořit buď sami nebo pomocí nástroje maildirmake (poskytuje ho řada balíků, např. balík maildrop).

Poštu je také možné ukládat jinam, např. do databáze. Tím se ale v seriálu zabývat nebudu.

### Vybírání pošty: POP3, IMAP4

V tuto chvíli již tušíte, jakým způsobem se pošta přepraví z jednoho počítače na druhý a kdo ji uloží do poštovní schránky. Zbývá tedy poslední krok – možnost si poštu vybrat. Vybírání pošty se realizuje prostřednictvím některého ze dvou možných protokolů, POP a IMAP. Přesněji pak POP verze 3 a IMAP verze 4. IMAP4 je obvykle používán tak, že klient k serveru zůstává připojen a stahuje zprávy nebo provádí jiné operace na požádání. U POP3 se obvykle klient připojuje, provede svou činnost a zase se odpojí.

IMAP4 umí oproti POP3 navíc záležitosti jako možnost se ke schránce připojit najednou z více klientů, stahovat pouze části e-mailu (přesněji jednotlivé MIME části), synchronizovat informace o stavu zpráv (např. zda-li byla zpráva přečtena či nikoliv) mezi jednotlivými klienty, atd. Jeho nevýhodou pak je nejspíše související vyšší komplexita protokolu a případné vyšší nároky na server.

Aby si mohl uživatel poštu stahovat, musí na serveru běžet POP3 nebo IMAP4 server (nebo obojí). To je ve světě GNU/Linuxu obvykle Dovecot nebo Courier.

### Přehled protokolů a jejich portů

Pro úplnost uvádím krátký přehled protokolů poštovních služeb a jejich odpovídající čísla portů:

- SMTP: port 25, SSL varianta SMTPS, port 465
- POP3: port 110, SSL varianta POP3S, port 995
- IMAP4: port 143, SSL varianta IMAPS, port 993

Dodám, že u všech tří protokolů je možné využít Transport Layer Security, tedy TLS. Pokud tuto možnost podporuje server, klient si může touto cestou na „nešifrovaném" portu vyžádat šifrování.

### MUA: Mail User Agent

Mail User Agent je klient využívaný uživatelem, který mu zpřístupňuje poštovní schránku a umožňuje mu poštu stahovat (a obvykle i odesílat). MUA může být v GNU/Linuxu řada grafických poštovních klientů jako Thunderbird, Claws-mail, Evolution, K-Mail, atd., ale také řada řádkových klientů (Mutt, Alpine, Sup, apod.). Samozřejmě, budete-li provozovat poštovní server pro veřejnost, budou vaši uživatelé patrně využívat i jiné poštovní klienty určené pro jiné operační systémy, a vy pak můžete řešit problémy spojené s chybami nebo specifickými vlastnostmi takových klientů.

#### *Webmail*

Mnoho lidí v dnešní době využívá přístup k poštovní schránce prostřednictvím webových klientů. V oblasti linuxových serverů a svobodného softwaru jsou dostupné přinejmenším dva webmaily, jednodušší Squirrel Mail a komplexnější Roundcube. Roundcube využívá AJAX, a jako takový se může řadit pod Web 2.0. Webmail je obvykle vázán na protokol IMAP – na serveru tedy musí být k dispozici IMAP server (stačí nešifrovaný na localhostu). Vazbu na SMTP server pro odesílání pošty asi nemusím příliš zdůrazňovat. Z bezpečnostních důvodů nicméně důrazně doporučuji chránit přístup k poštovní schránce pomocí SSL, na úrovni webového serveru.

### Spam a viry

Boj se spamem, popřípadě s viry, představuje patrně tu největší zátěž pro správce poštovního serveru. Zatímco poštovní server je možné jednou nastavit a nechat ho pracovat, spam nutí správce upravovat a vylaďovat nastavení za běhu, popřípadě reagovat na neobvyklé situace (e-mail bez odpovědi, odmítání příjmu e-mailu nějakým poštovním serverem, atd.).

Se spamem každý správce poštovního serveru nakládá po svém. Možností nastavení je nepřeberně a ne vždy jsou respektována příslušná RFC. Díky aktivnímu boji se spamem je v dnešní době obtížné provozovat třeba domácí poštovní server, a to nejenom pokud sdílíte rozsah sítě s řadou zavirovaných počítačů, které rozesílají spam.

Řada lidí dnes využívá nejrůznější freemaily, které mají různé politiky pro klasifikaci spamu. Čím větší služba, tím bývá obtížnější získat kontakt na správce a zjistit důvod, proč vaše e-maily končí klasifikovány jako spam. Tento problém se samozřejmě netýká jen freemailů.

#### *Antispam*

Základním prostředkem pro boj se spamem je antispamový filtr, který prohlíží e-maily a hledá typické znaky spamu. Každý znak má obvykle přiřazeno určité skóre. Po sečtení skóre všech znaků se výsledná hodnota porovná s prahovou hodnotou. Překročí-li skóre e-mailu prahovou hodnotu, je e-mail klasifikován jako spam. Co se s takovým e-mailem stane, závisí na správci. Může být zahozen, doručen do schránky s modifikovaným předmětem nebo doručen do speciální složky pro spam.

Typickým nástrojem tohoto typu je spamassassin. Kromě výše uvedeného umí využívat i Bayesův filtr, který využívá teorie pravděpodobnosti a dokáže se „učit". Před jeho použitím je dokonce velmi vhodné ho „naučit". Více o něm si můžete přečíst třeba v našem článku Bayesův filtr.

Stejně jako jiné nástroje pro boj se spamem může spamassassin klasifikovat i běžnou zprávu jako spam. Proto bývá vhodné jej pečlivě nastavit a po nějakou dobu sledovat, jak poštu klasifikuje.

#### *Antivirus*

Ve světě Linuxu neexistuje příliš velká motivace používat antivirus, s výjimkou provozování poštovního serveru. Implementace nějakého antiviru je vhodná, přinejmenším kvůli uživatelům využívajícím jiný operační systém než GNU/Linux.

Takovým antivirem může být např. clamav, který bývá součástí většiny distribucí.

#### *Restrikce na úrovni poštovního serveru*

Na úrovni poštovního serveru lze specifikovat řadu restrikcí. Je možné vynutit ohlášení klienta pomocí HELO/EHLO, je možné kontrolovat hodnotu MAIL FROM: a ujistit se, že je použito plně kvalifikované doménové jméno, popřípadě že je použito existující doménové jméno, apod. Problémem tohoto přístupu je, že tím můžete některé poštovní servery (nebo poštovní klienty) odříznout, a spolu s nimi odříznout i regulerní poštu.

#### *Blacklisty*

Veřejné blacklisty představují jistou kontroverzní metodu boje se spamem. Za blacklisty obvykle stojí nějaká organizace. Fungují tak, že určitou metodou sbírají určité IP adresy a prostřednictvím DNS dotazů umožňují poštovním serverům zjistit, zda-li se klient, který se právě připojil a hodlá předat zprávu, nachází či nenachází v daném blacklistu. Pokud se v něm nachází, poštovní servery pak zprávu obvykle odmítnou doručit (a vygenerují příslušnou chybovou hlášku), ale kontrolu vůči blacklistům je možné implementovat i jinam, např. do antispamu, a u e-mailu odeslaného z počítače v blacklistu pouze zvýšit skóre.

Blacklisty mohou obsahovat nejenom počítače rozesílající spam, ale také počítače, které se nachází v síti nějakého ISP (např. poskytovatele připojení pro domácnosti), nebo počítače, které se na seznam dostanou omylem.

Budete-li používat blacklisty, určitě se podívejte, jaké IP adresy se sbírají a jakou metodikou. Zároveň se také podívejte, co je třeba k tomu, aby mohl správce svůj server z blacklistu vyřadit. Blacklisty, které vyžadují zaplacení poplatku nebo mají přehnané požadavky, byste používat neměli, alespoň v rámci solidarity s kolegy správci jiných poštovních serverů.

Problém s blacklisty lze opět shrnout na možné odmítání regulerní pošty, popřípadě na obtížnost vyjmutí serveru z blacklistu.

Dlužno dodat, že existují i široké možnosti lokálních blacklistů, tedy blacklistů na úrovni vašeho serveru. Můžete odmítat poštu na základě nejrůznějších vlastních kritérií. Zde je výhodou, že nad tímto procesem máte plnou kontrolu.

***Greylisting***

Greylisting je z praktického hlediska velmi účinnou metodou, přitom jeho princip je velmi jednoduchý. V rámci něj se kontroluje tzv. triplet, tedy kombinace IP adresy doručovatele e-mailu, odesílatele a příjemce (dle SMTP). Byl-li tento triplet viděn dříve, server e-mail akceptuje. Pokud se objevil poprvé, server e-mail dočasně odmítne převzít (např. po dobu 5 minut). Teprve po opětovném doručení po uplynutí této doby jej převezme a daný triplet uloží do databáze.

Greylisting vychází z toho, že normální SMTP server by se měl pokusit zprávu doručit znovu, zatímco nástroje spammerů tak obvykle nečiní. Nevýhodou Greylistingu je související prodleva – ta může trvat různě dlouho, 15 minut i čtyři hodiny (nebo déle). Špatně nakonfigurované SMTP servery vám třeba nemusí daný e-mail znovu doručit vůbec – po prvním pokusu to vzdají. Může také dojít k jiným problémům v závislosti na okolnostech (viz odkazy pod článkem). Greylisting je však v dnešní době natolik používanou metodou, že by na ni *měly* SMTP servery brát ohled.

***Akce***

V mnoha situacích můžete rozhodnout, co se má stát s e-mailem, který nevyhověl některému pravidlu. Můžete jej odmítnout převzít, přesměrovat, zahodit nebo doručit do specifické složky, atd. Některé z těchto řešení jsou vhodné, jiné zcela katastrofální (např. tiché zahození e-mailu). Vždy se pokuste myslet na ty, kteří vám chtějí doručit regulerní poštu – jak zjistí, že pošta dorazila nebo ne, když e-mail zůstane bez reakce?

***Nepříjemné komplikace***

Spam představuje velký problém, který různí správci řeší různě. Díky němu se v doručování e-mailů objevují prodlevy (Greylisting), regulerní e-maily končí ve spamovém koši, pošta z určitých serverů je odmítána na základě blacklistů, přičemž s tím vším jako správci přijdete do styku. A to jak z pohledu konfigurátorů a pánů nad vlastním serverem, tak z pohledu SMTP serverů, které se snaží doručit poštu na jiný server, který se s vámi nechce bavit.

## Správa linuxového serveru: Nastavení DNS, SPF a Postfix

Po převážně teoretických dílech následuje díl, ve kterém se dozvíte, jak nainstalovat poštovní server Postfix a provést jeho základní nastavení. Článek také osvětlí nastavení DNS pro poštovní server a SPF, tedy Sender Policy Framework.

Minulé dva díly (první díl, druhý díl) se věnovaly převážně teorii, nicméně je vhodné si je projít před tímto článkem.

### Nastavení DNS

Předpokladem pro funkci poštovního serveru je správně nastavené a fungující DNS. U příslušných domén, kterým chcete dělat pošťáka, byste měli mít nastaveny MX záznamy (pokud je nastaveny nemáte, využijí poštovní servery pro doručování pošty A záznam pro danou doménu). Pomocí MX záznamu můžete specifikovat nejenom jeden nebo více poštovních serverů pro danou doménu, ale také jejich priority (nižší číslo značí vyšší prioritu). Tímto způsobem si můžete vytvářet záložní poštovní servery pro případ, že vám ten hlavní přestane na nějakou dobu fungovat. Kromě MX záznamu bývá vhodné definovat reverzní záznam pro IP adresu poštovního serveru tak, aby odpovídal jeho plně kvalifikovanému doménovému jménu serveru. Bez toho se vystavujete možnosti, že vaši poštu budou některé servery penalizovat (a možná klasifikovat jako spam) nebo odmítat.

Existují různá dodatečná rozšíření (zpravidla využívající DNS), která pomáhají garantovat původ pošty a omezit šíření spamu. Mezi ty patří zejména SPF, popřípadě DKIM. Jejich implementace není nutná, ale přinejmenším SPF bych implementovat doporučil. Nevyžaduje to žádné úpravy konfigurace, pouze nastavení DNS záznamu.

### *SPF*

SPF je zkratka pro Sender Policy Framework. Představuje mechanismus, jímž můžete definovat servery, které mají oprávnění odesílat poštu z vaší domény. Díky tomu mohou ostatní poštovní servery automaticky odmítat poštu, která jako odesílatele (buď v MAIL FROM: v rámci SMTP komunikace, nebo v poli From: v hlavičce e-mailu) specifikuje uživatele z vaší domény, ale posílá ji neautorizovaný server (nejspíše spammer). Nasazením SPF si také můžete pomoci u antispamu některých poštovních serverů, které mohou vašemu mailu přidat nějaké bodíky k dobru. SPF nastavení se umisťuje do TXT záznamu pro danou doménu.

Informace o SPF můžete nalézt na webu openspf.org nebo na Wikipedii. Nejprostší nastavení může vypadat takto:

```
v=spf1 a mx -all
```

Záznam je uvozen specifikací verze SPF, tedy v=spf1, následuje specifikace všech počítačů, které jsou nebo nejsou oprávněny odesílat poštu pro danou doménu. Zde je specifikován počítač odpovídající A záznamu domény (identifikátor a) a všechny počítače specifikované v MX záznamech pro danou doménu (identifikátor mx). Ostatní počítače (all) jsou považovány za neautorizované a jejich pošta by se měla odmítat.

U každého prvku můžete znaménkem určit, jak je či není oprávněn odesílat poštu (nespecifikujete-li žádné znaménko, předpokládá se, že daný prvek je povolen):

- `+` odesílání pošty je povoleno, zprávy z těchto počítačů by měly být přijaty
- `-` odesílání pošty je zakázáno, zprávy z těchto počítačů by měly být odmítnuty
- `~` odesílání pošty je polozakázáno (softfail) – taková pošta by měla projít, ale současně by měla být adekvátně označena
- `?` neutrální, pro daný prvek není definována žádná politika

Syntax SPF je velmi dobře popsána na openspf.org.

### Instalace Postfixu

Jako téměř vždy, stačí nainstalovat jediný balíček, shodný s názvem nástroje, který chcete nainstalovat. V Debianu byste Postfix nainstalovali takto:

```
aptitude install postfix
```

Používáte-li standardní, čistou instalaci Debianu, nejspíše budete mírně překvapeni tím, že instalace Postfixu vyžaduje odstranění některých balíčků, konkrétně pak Eximu. Exim je v Debianu výchozím poštovním serverem a nemůže fungovat vedle Postfixu, je tedy třeba jej odstranit.

Během instalace budete dotázáni na variantu nastavení poštovního serveru. V úvahu přichází následující:

- Internetový počítač – poštovní server je připojen k internetu, přijímá a odesílá poštu sám
- Internet se smarthostem – poštovní server je připojen k internetu, poštu přijímá přes SMTP, odesílá ji však přes jiný počítač (např. centrální mailserver ve firmě, mailserver vašeho ISP apod.)
- Satelitní systém – poštovní server běží pouze na localhostu (není tedy možné na něj doručovat poštu zvenčí) a veškerou poštu odesílá přes jiný počítač
- Pouze tento počítač – pouze lokální doručování, server není připojen k internetu

Budete také dotázáni na plně kvalifikované doménové jméno vašeho poštovního serveru (FQDN) a některé další otázky v závislosti na tom, které nastavení jste si zvolili. Tyto parametry jsou probrány níže.

Tato nastavení máte kdykoli možnost změnit buď ruční úpravou konfiguračních souborů, nebo opětovným spuštěním konfigurátoru (debconf), což můžete provést příkazem:

```
dpkg-reconfigure postfix
```

### Základní nastavení Postfixu

Konfigurace Postfixu je uložena v `/etc/postfix`, přičemž hlavní konfigurační soubor je `/etc/postfix/main.cf`. V dalších dílech vás provedu pokročilejší konfigurací, během které se patrně dostaneme i k řadě jiných souborů, ale pro základní nastavení vám stačí výše uvedený hlavní konfigurační soubor.

Základní nastavení Postfixu může vypadat nějak takto:

```
myhostname = mail.example.net
mydomain = example.net
myorigin = /etc/mailname
mydestination = example.net, example.org, localhost.localdomain, localhost
relay_domains = $mydestination
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
inet_interfaces = all
inet_protocols = all
```

Jednotlivé volby by bylo dobré osvětlit jednu po druhé:

```
myhostname = mail.example.net
```

Toto by mělo být plně kvalifikované doménové jméno (FQDN) vašeho poštovního serveru, tedy v příkladě výše mail.example.net.

```
mydomain = example.net
```

Proměnná mydomain by měla obsahovat doménovou část vašeho myhostname. V případě mail.example.netto bude example.net.

```
myorigin = /etc/mailname
```

Doménové jméno, které bude použito pro poštu, která je odesílána lokálně. Pokud tedy jako uživatel přihlášený na server třeba přes SSH odešlete e-mail z příkazové řádky, např. takto:

```
echo 'Ahoj, světe.' | mail -s 'Pozdrav' prijemce@mail.example.eu
```

bude se v takovém případě jako odesílatel brát uzivatel@myorigin, kde myorigin je hodnota dané proměnné, tj. v příkladu výše to bude hodnota obsažená v souboru /etc/mailname.

```
mydestination = example.net, example.org, localhost.localdomain, localhost
```

mydestination obsahuje seznam domén, pro které bude Postfix poštu přijímat a doručovat do poštovních schránek. Pro tyto domény bude poštovní server cílovým bodem. Všimněte si, že v tomto případě je specifikováno více domén, konkrétně dvě (example.net a example.org).

```
relay_domains = $mydestination
```

Tato proměnná určuje, pro které domény se bude pošta přeposílat, resp. předávat (relay). Můžete ji ponechat prázdnou, nebo sem dosadit hodnotu proměnné $mydestination, což lze provést způsobem naznačeným výše. Toto je poměrně zajímavá vlastnost nastavení Postfixu – jako součást hodnoty nějaké proměnné můžete použít hodnotu jiné proměnné.

Je klíčové, aby váš poštovní server nefungoval jako „open relay“, tedy server, který ochotně převezme a doručuje jakoukoliv poštu, která k němu přijde a není v mydestination. Takový server se velmi rychle ocitne v blacklistech. Na webu je dostupná množina nástrojů, které vám umožní svůj server otestovat, a já vám důrazně doporučuji některý z těchto nástrojů použít (hledejte „open relay test“), protože pokud to neuděláte vy, spammeři to určitě zkusí.

```
relayhost =
```

Tato proměnná umožňuje specifikovat cizí poštovní server, kterému bude váš server předávat veškerou poštu, která nebude doručována lokálně. Pošta směřující „ven“ tedy nebude při použití tohoto parametru doručována přímo vaším poštovním serverem, ale odeslána na počítač specifikovaný jako parametr této proměnné.

```
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
```

Počítače specifikované v mynetworks smějí přes váš server odesílat poštu. Je velmi vhodné zde specifikovat pouze místní rozsahy nebo opravdu důvěryhodné sítě.

```
inet_interfaces = all
```

Tato proměnná určuje, na kterých rozhraních (resp. IP adresách a rozsazích) bude Postfix naslouchat. Pokud zde specifikujete 127.0.0.1, váš poštovní server bude k dispozici pouze lokálně a nebude naslouchat na síti. To se hodí, pokud daný počítač neslouží pro příjem pošty. Hodnota all zajistí, že Postfix bude naslouchat na všech dostupných IP adresách.

```
inet_protocols = all
```

Zde můžete specifikovat, jaké verze IP protokolu bude Postfix používat. V úvahu přichází ipv4, ipv6 neboall. Jsou-li použity oba protokoly, mějte na paměti, že IPv6 v tomto případě dostává přednost, tj. pokud bude moci Postfix doručit poštu na IPv4 i IPv6 adresu, zkusí nejprve IPv6 adresu.

Tolik k základnímu nastavení. Po změně nastavení je vhodné server restartovat:

```
/etc/init.d/postfix restart
```

Funkčnost serveru můžete otestovat pomocí znalostí z minulých dílů:

```
telnet server.example.org 25
```

Připomínám, že byste měli určitě otestovat, zda váš server není open relay (viz výše).

Postfix je velmi komplexní nástroj a má široké možnosti nastavení. V odkazech pod článkem naleznete jak odkazy na dokumentaci Postfixu, tak odkazy na wiki různých distribucí, ve kterých existují záznamy o Postfixu.

Tím bych tento díl ukončil. Příště vás provedu podrobnějším nastavením Postfixu.

## Správa linuxového serveru: Postconf, řízení fronty, greylisting

Tento díl se zabývá nástrojem postconf, který vám usnadňuje nastavení Postfixu, dále řízením fronty, provozem Postfixu na více portech současně a greylistingem, jednou z nejúčinnějších taktik pro boj se spamem.

Na úvod připomenu první dva díly (první díl, druhý díl) o poštovním serveru, které se převážně zabývaly teorií, stejně jako minulý díl, který se věnoval základnímu nastavení Postfixu a související nastavení DNS a SPF. Pokud jste si je neprošli, doporučuji to udělat.

### Pár slov o nastavení Postfixu a nástroji postconf

Hlavní konfigurační soubor Postfixu, /etc/postfix/main.cf obsahuje (alespoň v případě Debianu) relativně málo voleb a minimum komentářů. Pokud nahlédnete do dokumentace, zjistíte, že Postfix má nepřeberné množství nejrůznějších voleb. Přirozeně, každá z nich má nějaké výchozí nastavení, takže pokud její nastavení v main.cf neupravíte, použije se výchozí hodnota. Proto může být konfigurace Postfixu relativně strohá – například pouze několikařádková. Konfiguraci Postfixu vám může podstatně ulehčit nástroj postconf, který je schopen s konfigurací manipulovat a vypisovat ji různým způsobemZavoláním postconf bez parametrů docílíte vypsání aktuálního nastavení. Hledáte-li něco konkrétního, použijte nástroj grep. Jedny z typických pokročilejších nastavení jsou nastavení nejrůznějších omezení, která si pomocí grepu můžete nechat elegantně vypsat:

```
postconf | grep restrictions
```

Nástroj postconf však umí nejenom to. V některém případě se vám hodí mít možnost vypsat výchozí hodnoty, nikoliv ty aktuálně nastavené. K tomu slouží přepínač -d (d jako *default*). Porovnejte:

```
server:~# postconf -d smtpd_use_tls
smtpd_use_tls = no
server:~# postconf smtpd_use_tls
smtpd_use_tls = yes
```

Pokud tedy někdy budete potřebovat rychle zjistit výchozí hodnotu pro danou volbu, hodí se vám parametr -d. Všimněte si také, že pokud se jedná o jednu konkrétní volbu, můžete ji zadat nástroji postconf přímo a on vám ji vyhledá a vypíše. V tomto případě tedy nepotřebujete grep. Postconf vám také umožňuje provádět editace konfiguračního souboru. Chcete-li tedy např. změnitmyhostname, můžete to provést kromě ruční editace takto:

```
postconf -e myhostname=mujnovyserver.example.org
```

Výše uvedený příkaz změní hodnotu v konfiguračním souboru, je-li tam definovaná, pokud ne, zapíše ji tam i s požadovanou hodnotou. Přirozeně, změny se projeví až při znovunačtení konfigurace Postfixu:

```
/etc/init.d/postfix reload
```

Na závěr přijde o nástroji postconf to nejdůležitější, a sice parametr -n. Ten je schopen vypsat hodnoty pouze těch parametrů, které se liší od výchozích hodnot. Je to de facto vaše unikátní konfigurace Postfixu:

```
postconf -n
```

Dokumentace jednotlivých voleb nastavení Postfixu je k dispozici v manuálových stránkách:

```
man 5 postconf
```

Zadat číslo sekce je zde nutné, bez něj se zobrazí dokumentace k samotnému nástroji postconf. Připomínám také možnost vyhledávat v manuálových stránkách (následující dva příkazy jsou ekvivalentní):

```
man -k postfix
apropos postfix
```

### Řízení fronty Postfixu

Ne všechny maily se nutně musí podařit doručit okamžitě, a to nikoliv nezbytně proto, že je cílový poštovní server nedostupný. Pokud jste četli předchozí díly, možná si vzpomínáte na techniku zvanou greylisting, která spočívá v odmítání doručení po určitou dobu. Poštovní server se v případě nedostupnosti cílového serveru podívá, jestli nejsou k dispozici nějaké jiné poštovní servery v MX záznamech pro danou doménu. Pokud ne, e-mail zůstane ve frontě a Postfix se ho časem pokusí znovu doručit. Po určité době (5 dní ve výchozím stavu) to vzdá úplně a pošle odesílateli zprávu o nedoručení. Tuto dobu je možné ovlivnit konfigurační volbou maximal_queue_lifetime (výchozí hodnota je „5d" (d jako *day*, tedy den). Frontu jako takovou můžete řídit pomocí nástroje postqueue. Vypsání e-mailů, které jsou ve frontě, docílíte příkazem:

```
postqueue -p
```

Chcete-li se pokusit frontu vyprázdnit, tedy nařídit Postfixu, aby se pokusil obsah fronty doručit ihned, použijte parametr -f:

```
postqueue -f
```

### Konfigurační soubor master.cf a Postfix na více portech

Postfix není jeden velký démon, ale hromada provázaných služeb, které jsou volány a reagují na různé podněty. Tento konfigurační soubor definuje právě tyto služby. Změny v tomto souboru jsou časté např. při nasazení služeb typu Amavis, které kontrolují obsah zpráv (antispam, antivirus). Úpravou tohoto souboru můžete kupříkladu nechat Postfix naslouchat na více portech, což se hodí v situaci, kdy vaši uživatelé chtějí odesílat poštu přes váš server, ale jejich poskytovatel připojení blokuje odchozí komunikaci na port 25. Pokud byste chtěli, aby Postfix naslouchal kromě portu 25 také na portu 250, přidali byste do tohoto souboru následující řádku:

```
250 inet n - - - - smtpd
```

Tato řádka definuje novou síťovou službu (klíčové slovo inet) běžící na portu 250 a příkaz spouštějící danou službu je smtpd, což odpovídá SMTP serveru Postfixu.

Podrobnější dokumentaci k tomuto konfiguračnímu souboru naleznete v manuálových stránkách:

```
man 5 master
```

### Greylisting

Greylisting je v boji se spamem jednou z nejúčinnějších metod, i když skýtá jistá rizika a nevýhody. Hlavním problémem je zpoždění doručení pošty, které vyvolává. To se může v praxi pohybovat od několika minut až po hodiny. Některé špatně nastavené poštovní servery nemusí být schopny poštu doručit vůbec (to by měly být ale opravdu výjimky).Greylisting byl zmíněn v tomto dílu, takže jen velmi stručně – tato technika spočívá v tom, že poštovní server odmítá přijmout e-mail z neznámého zdroje po určitou dobu. Standardně by se poštovní servery měly pokusit -e-mail doručit znovu, až se jim to nakonec podaří, zatímco spammeři obvykle e-maily dvakrát neposílají.Démonů zajišťujících tuto funkcionalitu existuje více. Patrně nejčastěji se setkáte s nástrojem Postgrey, napsaným v Perlu a určeným pro Postfix. Nainstalujete jej jako obvykle, pomocí správce balíčků:

```
aptitude install postgrey
```

Démon se nastavuje prostřednictvím voleb předaných na příkazové řádce. Tyto volby je možné upravit v souboru /etc/default/postgrey, konkrétně pak v proměnné POSTGREY_OPTS, která vypadá ve výchozím stavu takto:

```
POSTGREY_OPTS="--inet=10023"
```

Kromě zde nastaveného portu, kde služba běží, můžete nastavit zejména dobu, po kterou bude Postgrey odmítat e-mail z neznámého zdroje. Výchozí hodnota je 300 sekund, tedy pět minut. Pokud byste tuto lhůtu chtěli snížit na minutu, provedli byste to pomocí volby --delay, takto:

```
POSTGREY_OPTS="--delay=60 --inet=10023"
```

Žádný antispam by neměl postrádat whitelisting, tedy možnost některé klienty propustit bez kontroly (v případě greylistingu bez „ochranné lhůty"). Postgrey má k dispozici dva konfigurační soubory určené pro whitelisting:

- /etc/postgrey/whitelist_clients umožní propustit poštu pocházející z určitých zdrojů (domén či IP adres a rozsahů)
- /etc/postgrey/whitelist_recipients umožní propustit poštu doručovanou konkrétnímu příjemci (typicky např. postmaster)

Po úpravě konfigurace restartujte Postgrey:

```
/etc/init.d/postgrey restart
```

Integrace s Postfixem se provádí přes volbu smtpd_recipient_restrictions, která umožňuje klást požadavky na příjemce a odmítat poštu, která jim nevyhovuje. Více o restrikcích se dozvíte v příštím díle. Výchozí hodnotou této volby je řetězec dvou parametrů oddělených čárkou:

- permit_mynetworks – sítě, které Postfix spravuje, nebudou omezeny
- reject_unauth_destination – odmítne požadavek, pokud není Postfix finálním příjemcem zprávy nebo není nastaven tak, aby zprávu přeposílal dál

Postgrey zařaďte na konec této „fronty" pomocí check_policy_service, která provede kontrolu pomocí specifikované služby, takto:

```
smtpd_recipient_restrictions = permit_mynetworks,
```

```
reject_unauth_destination,
check_policy_service inet:127.0.0.1:10023
```

Následně zbývá donutit Postfix znovu načíst konfiguraci:

```
/etc/init.d/postfix reload
```

Postgrey má přirozeně své alternativy, u vytíženějšího serveru se může hodit, pokud se známé triplety ukládají do databáze (MySQL, PostgreSQL, atd.). K tomu vám může posloužit sqlgrey. A tím bych tento díl ukončil.

## Správa linuxového serveru: Implementace DKIM

DKIM je mechanismus pro zvýšení důvěryhodnosti e-mailů odesílaných z vašeho poštovního serveru. Jak funguje a jak jej zprovoznit, to se dozvíte v tomto dílu.

Připomínám, že poštovnímu serveru a otázkám s ním spojeným byly věnovány poslední čtyři díly seriálu. Pokud jste je nečetli, určitě se na ně podívejte – první dva díly (první díl, druhý díl) se věnovaly převážně teorii, třetí díl tvořil praktický úvod do nastavení Postfixu, DNS a SPF, techniky určená ke zvýšení důvěryhodnosti e-mailů, na což naváže dnešní díl tématem DKIM, a konečněčtvrtý díl, který se věnoval nastavení Postfixu obecně a antispamové technice zvané greylisting.

### DKIM

DKIM je zkratka pro DomainKeys Identified Mail, tedy pošta identifikovaná pomocí doménových klíčů. Jedná se o techniku ověření původu e-mailu (nebo, chcete-li, techniku zaručení se za odeslaný e-mail) pomocí asymetrického šifrování.

#### *Asymetrické šifrování*

Symetrické šifrování asi každý zná, máte algoritmus, klíč a nešifrovaná data. Šifrovacímu algoritmu předáte klíč a nešifrovaná data, načež on vám vyprodukuje zašifrovaná data. K dešifrování použijete stejný algoritmus (v módu dešifrování) a stejný klíč.

Asymetrické šifrování nevyužívá jeden klíč, ale dva klíče (přesněji pár klíčů), jejichž zásadní vlastností je to, že co zašifrujete jedním klíčem, rozšifrujete pouze tím druhým. Asymetrického šifrování se využívá v mnoha oblastech, včetně kupříkladu oblasti digitálních podpisů.

Digitální podepisování funguje tak, že si vygenerujete pár klíčů, z toho jeden (veřejný) klíč zveřejníte a druhý (soukromý) si chráníte. Z dat, která podepisujete, se spočítá hash, který se následně zašifruje vaším soukromým klíčem. Všichni, kteří mají k dispozici váš veřejný klíč, mohou daný hash dešifrovat, spočítat si hash podepsaných dat a jejich porovnáním zjistit, jestli jsou data stejná nebo zda nebyla pozměněna. Tyto postupy jsou samozřejmě zautomatizované.

#### *Fungování DKIM*

DKIM je v podstatě systém digitálního podepisování zpráv na úrovni serveru, a to pro každou z obsluhovaných domén. Ověřením tohoto podpisu si může kterýkoliv jiný poštovní server ověřit, že zpráva pochází skutečně ze serveru autorizovaného pro odesílání pošty z dané domény. Tato technika, spolu se SPF, umožňuje přidat vaší poště na důvěryhodnosti. Samozřejmě pouze u těch serverů, které provádí příslušná ověřování (SPF či DKIM).

#### *Zprovoznění DKIM*

Zprovoznění DKIM představuje následující kroky:

1. instalace DKIM filtru
2. vygenerování klíčů pro domény
3. úprava DNS záznamů
4. nastavení DKIM filtru
5. nastavení Postfixu

Instalace DKIM filtru

Samotná instalace představuje instalaci balíku dkim-filter, což v Debianu provedete takto:

```
aptitude install dkim-filter
```

Vygenerování doménových klíčů

Jelikož DKIM nemá v Debianu připraveno úložiště klíčů v /etc, je třeba nějaké vytvořit, např.:

```
mkdir -p /etc/dkim/keys
cd /etc/dkim/keys
```

Následně vygenerujte klíče pro každou doménu:

```
mkdir -p /etc/dkim/keys/example.com
cd /etc/dkim/keys/example.com
dkim-genkey -b 1024 -d example.com
```

Parametr -b udává délku generovaného RSA klíče, přípustné hodnoty jsou mezi 512 a 2048 bitů. Doporučená hodnota je 1024. Parametr -d udává doménu, pro kterou má být pár klíčů generovaný.

Parametr -s určuje tzv. selektor, který vám umožňuje přiřadit více klíčů jedné doméně, pokaždé s odlišným selektorem). Výchozím selektorem je *default*. Klíč určený k podepisování (soukromý klíč) se uloží do souboru selektor.private v aktuálním adresáři, veřejný klíč pak do souboru selektor.txt. Není tedy od věci pro každou doménu vyhradit specifický podadresář, tak, jak je naznačeno výše.

Úprava DNS záznamů

Aby si ostatní poštovní servery mohly podpisy ověřovat, je třeba umístit veřejné klíče na nějaké veřejně dostupné a důvěryhodné místo. Stejně jako v případě SPF i DKIM využívá DNS, konkrétně pak TXT záznamů. Samotný DNS záznam nemusíte nijak tvořit, stačí jej zkopírovat ze souboru selektor.txt. Pro představu, příslušný záznam vypadá např. takto:

```
default._domainkey IN TXT "v=DKIM1; g=*; k=rsa;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC65bMhv0ACIGOzAT4uXbGC3dKo5opmtnnIASUFbsmA8qiFjjUO+fZQrOI3UiUx0laPANnarGn+caTv7jdJAS+dhpCIvA4zUixIDk0opm+Xv6IeSZ9SyVLGkE7+Jtos0He5+h3lVRz/GqbJLAR5yvgXyn7nzC7cGebjfOgCbeBADwIDAQAB" ; ----- DKIM default for example.com
```

Povšimněte si selektoru, který má hodnotu *default* (hned na začátku řádku).

Nastavení DKIM filtru

Nejprve je třeba démonovi říci, jaké domény má obsluhovat. Toho docílíte vytvořením souboru s vhodně formátovaným seznamem, který může vypadat třeba takto:

```
*@example.cz:example.cz:/etc/dkim/keys/example.cz/default.private
*@example.com:example.com:/etc/dkim/keys/example.com/default.private
```

Jak je vidět, na každém řádku je záznam pro jeden klíč, parametry jsou celkem tři a jsou odděleny dvojtečkou. Prvním parametrem je vzor odesílatele (hvězdička funguje jako zástupný znak pro nula nebo více znaků), druhým pak doména, pro kterou bude podepisováno, a třetím parametrem je umístění souboru s klíčem. Kam tento konfigurační soubor umístíte nebo jak ho pojmenujete, je v zásadě jedno. V souladu s příkladem výše by mohl být umístěn v /etc/dkim/dkim-keys.conf.

Podstatné je, abyste o tomto souboru řekli DKIM filtru – upravte tedy soubor /etc/dkim-filter.conf, kam přidejte následující řádek:

```
KeyList /etc/dkim/dkim-keys.conf
```

Pokud jste výše uvedený soubor se seznamem domén pojmenovali jinak, upravte parametr direktivyKeyList tak, aby odpovídal absolutní cestě k tomuto souboru.

Dalším krokem je nastavení portu, na kterém bude DKIM filtr naslouchat. Upravte soubor/etc/default/dkim-filter a vložte nebo odkomentujte a upravte stávající záznam podle následujícího vzoru:

```
SOCKET="inet:8891@localhost"
```

Tento řádek, jak je asi jasné, nastaví DKIM k poslechu na portu 8891. Chcete-li použít jiný port, samozřejmě můžete.

Následně DKIM filtr restartujte:

```
/etc/init.d/dkim-filter restart
```

A přesvědčte se, že běží, a to na správném portu:

```
netstat -tlpn | grep 8891
```

Nastavení Postfixu

Do /etc/postfix/main.cf umístěte následující řádky:

```
milter_default_action = accept
milter_protocol = 2
smtpd_milters = inet:localhost:8891
non_smtpd_milters = inet:localhost:8891
```

První řádek nastavuje chování Postfixu v případě, že je filtr nedostupný: accept zařídí, že se e-mail zpracuje, jako kdyby nebyl nastaven žádný filtr. Druhý řádek nastavuje typ protokolu, zbylé dva řádky definují seznamy filtrů pro poštu, která dorazí buď pomocí SMTP démona, nebo jiným způsobem.

Následně řekněte Postfixu, aby znovu načetl konfiguraci:

```
/etc/init.d/postfix reload
```

***Otestování funkce DKIM***

Nejjednodušším způsobem otestování funkce DKIM je odeslat e-mail s odesílatelem odpovídajícím specifikovanému vzoru (nejspíše příslušné doméně). To můžete provést, jste-li přes SSH připojeni k serveru, třeba takto:

```
telnet localhost 25
```

Tím se připojíte na místní poštovní server. Dle SMTP protokolu byste měli zadat následující příkazy:

```
EHLO localhost
MAIL FROM: root@example.com
RCPT TO: root@localhost
DATA
Subject: DKIM test

.
```

Následně zkontrolujte poštovní schránku, kam vám chodí pošta pro roota, a ověřte, že se v e-mailu nachází DKIM podpis – ten může vypadat třeba takto:

```
DKIM-Signature: v=1; a=rsa-sha256; c=simple/simple; d=example.com;
s=example.com.private; t=1335887965;
bh=frcCV1k9oG9oKj3dpUqdJg1PxRT2RSN/XKdLCPjaYaY=;
h=Message-Id:Date:From:To;
b=O+LTeP0rfmUn9nnvsIyXY6uiaGi8wPt4/6myFoxWmeY8ivHMNdlbi+lZvQ2EkhaUG
f8T8zbzYY5/4GQdMF+F/N5mzcnFi8XmdgIoVWggpXnARuh+M1p4ui+VbUoi6DJGP/w
Grj7F78QdLXrFb8D3cV3q+yYFJ4RQpjMavhh/gxk=
```

Tento test vám ale neřekne, jestli jsou jiné servery schopny za pomoci doménových záznamů tento podpis ověřit. Nejjednodušší je asi použít některý z mnoha online nástrojů, které k tomuto účelu slouží. Ty naleznete buď pomocí vyhledávače, nebo v komentářích pod jedním z odkazovaných článků.

## Správa linuxového serveru: Aliasy a záložní poštovní server

Tento díl osvětlí problematiku aliasů a prozradí vám, jak nastavit záložní poštovní server, který bude vaši poštu přijímat a uchovávat, zatímco bude váš hlavní poštovní server mimo provoz.

### Aliasy

Aliasy představují metodu, jak více e-mailových adres směrovat do jediné schránky. Aliasy jsou definované v souboru /etc/aliases, kde jsou již připraveny výchozí aliasy (kupříkladu, dle RFC by každý SMTP server měl přijímat poštu směrovanou na postmaster@domena.tld a jeden z aliasů přeposílá tuto poštu rootovi). Syntax tohoto konfiguračního souboru je velmi jednoduchá:

```
alias: schranka
```

Tento záznam doručí poštu směřující na alias@domena.tld do schránky uživatele schranka. Specifikovat můžete ale i více příjemců, takže pokud třeba server spravuje více správců, můžete poštu pro roota posílat každému z nich:

```
root: michal@example.org, pepa@example.org, radek@example.org
```

Jste-li zvyklí po úpravě konfiguračního souboru restartovat server a očekávat, že se změna projeví, v tomto případě vás zaskočí mírně neintuitivní chování Postfixu. Pokud totiž provedete úpravu /etc/aliases a pouze restartujete poštovní server, změny se neprojeví. Postfix totiž nepracuje přímo s konfiguračním souborem /etc/aliases, ale s databázovým souborem (/etc/aliases.db), který se z tohoto souboru generuje. A to nikoliv automaticky, ale pouze na pokyn správce, příkazem:

```
newaliases
```

Teprve poté sdělte Postfixu, že si má znovu načíst konfiguraci:

```
service postfix reload
```

Zde je třeba dodat, že řada seznamů hodnot či pravidel se v Postfixu vytváří a spravuje tímto způsobem, tj. ze souboru si musíte nechat vygenerovat příslušný binární databázový soubor, a to příkazem postmap. Příkaz newaliases slouží pouze pro opětovné vygenerování databáze aliasů, kdežto postmap vám umožňuje vygenerovat databázi z libovolného (vhodně formátovaného) souboru:

```
postmap /etc/aliases
```

Nástroj newaliases de facto provede výše zmíněný příkaz.

### Záložní poštovní server

Záložní poštovní server se hodí v situaci, kdy váš hlavní poštovní server vypadne. Přirozeně, rozumně nastavené poštovní servery by neměly doručování na váš poštovní server rovnou vzdát, měly by si poštu pro váš server uložit a zkoušet ji doručit, dokud neuplyne timeout (výchozí timeout v Postfixu je 5 dní).

Záložní servery se specifikují pomocí MX záznamů DNS. Každý poštovní server má přiřazenou prioritu, která je daná kladným přirozeným číslem, kde vyšší číslo značí **nižší** prioritu.

Tvorba záložního serveru může být velmi jednoduchá, ale skýtá jedno velké ale. Samotnou funkčnost záložního serveru lze zrealizovat přidáním domén, pro které chcete dělat záložní server, do proměnnérelay_domains v /etc/postfix/main.cf:

```
relay_domains = example.com, example.org
```

Přirozeně, tyto domény nesmí být specifikovány v mydestination nebo jinde, kde se nastavuje nějaký typ místního doručování.

A nyní ono veliké ale. Za předpokladu, že je hodnota proměnné relay_recipient_maps prázdná, Postfix bude normálně přijímat veškerou poštu, která bude určena pro výše nastavené domény. Včetně pošty pro uživatele, kteří na hlavním poštovním serveru neexistují. Kdyby se totéž odehrálo na hlavním serveru, ten by prostě ohlásil, že daný uživatel neexistuje a odmítl poštu převzít. Záložní server v této konfiguraci však poštu přijme, a až bude hlavní server opět k dispozici, vygenerují se oznámení o nedoručení, která se rozešlou všem chudákům, které spammeři vyplnili do pole From.

Co s tím dělat? Kromě možnosti, že záložní poštovní server vůbec nebudete používat, zbývá volbarelay_recipient_maps (v souboru /etc/postfix/main.cf), která vám umožňuje specifikovat platné uživatele domén, pro které váš server slouží jako záložní:

```
relay_recipient_maps = hash:/etc/postfix/relay_recipients
```

Následně vytvořte soubor /etc/postfix/relay_recipients a naplňte ho platnými adresami:

```
uzivatel1@example.com x
uzivatel1@example.org x
uzivatel2@example.com x
uzivatel2@example.org x
```

Pokud takový seznam nechcete nebo nemůžete vytvořit a výše popsaný problém vám nevadí, ponechterelay_recipient_maps prázdné, což zapříčiní, že Postfix bude přijímat všechnu poštu. Je-li tato volba neprázdná, bude Postfix odmítat všechno kromě položek v daném seznamu.

### Unixové a virtuální poštovní účty

Standardní nastavení Postfixu, které jsem vám ukázal v jednom z minulých dílů, doručuje poštu všem lokálním unixovým účtům. Máte-li tedy uživatele michal a poštovní server na doméně example.com, bude se pošta směřující na michal@example.com doručovat do poštovní schránky uživatele michal. Obsluhuje-li server více domén, bude pošta ze všech domén směřovat na jediný unixový účet (tzn. pošta směřující namichal@example.cz, michal@example.org a michal@example.com bude doručena do schránky uživatelemichal).

U jedné domény nebo u osobního serveru je tato situace většinou tolerovatelná. Potřebujete-li však doručit poštu směřující na info@example.com a info@example.org do dvou různých schránek, musí přijít na řadu jiné řešení. Možnosti jsou zde v podstatě dvě:

- virtuální aliasy (směrovat specifické e-mailové adresy na různé unixové účty)
- vytvořit pro uživatele pošty virtuální účty, nesvázané s unixovými

Druhá z variant se často realizuje ve spolupráci s nějakou databází (MySQL, PostgreSQL), kde se uchovávají informace o účtech, zatímco pošta se směruje buď do souborů, nebo také do databáze. Jedná se o komplexní řešení, které bude probráno v jednom z následujících dílů.

První varianta je oproti druhé velmi jednoduchá. Přidejte do /etc/postfix/main.cf následující řádky:

```
virtual_alias_domains = example.org example.com
virtual_alias_maps = hash:/etc/postfix/virtual
```

Do proměnné virtual_alias_domains vložte seznam domén, pro které chcete tvořit virtuální aliasy, oddělovačem je zde mezera. **Pozor!** V žádném případě nesmíte stejnou doménu uvést vvirtual_alias_domains a v mydestination! Doména může být vždy jen v jedné z těchto proměnných.

Proměnná virtual_alias_maps obsahuje odkaz na soubor /etc/postfix/virtual, který nyní musíte vytvořit, a to s obsahem podobným tomuto:

```
info@example.org michal
info@example.com radek
postmaster@example.com root
postmaster@example.org root
@example.com domenovykos
```

Zde je asi na první pohled vidět, co je jak nastaveno. Maily směřující na info@example.org budou doručeny uživateli michal, zatímco maily směřující na info@example.com budou doručeny uživateli radek. Je zde definován alias pro postmastera, který v případě obou domén doručí poštu rootovi. Příklad obsahuje i doménový koš, tedy zachytávání pošty pro všechny neexistující uživatele, a to v tomto případě do schránky uživatele domenovykos.

Nyní je třeba vygenerovat databázový soubor z /etc/postfix/virtual, což můžete provést nástrojempostmap:

```
postmap /etc/postfix/virtual
```

Následně nechte Postfix znovu načíst konfiguraci:

```
service postfix reload
```

Doporučuji poté toto nastavení otestovat zasláním testovacích e-mailů. To, jak byla pošta doručována a kam, si můžete ověřit z logů – v Debianu je to /var/log/mail.log.

## Správa linuxového serveru: Restrictions Postfixu, blacklisty a whitelisty

Dnešní díl pokračuje v tématu boje se spamem, tentokrát pomocí omezení (restrictions) Postfixu a blacklistů/whitelistů.

Stejně jako veškeré ostatní antispamové metody (a metody řízení přístupu), které tu byly a budou představeny, představují i „restrictions" v Postfixu dvousečnou zbraň. Na jednu stranu pomáhají v odmítání nechtěné a nevyžádané pošty, ale stejně tak mohou zabránit v přijetí regulérního mailu – stačí ne zcela vhodně nastavený server. Vždy je třeba zvážit přínos versus náklady, kde přínosem je vliv na příjem a propouštění spamu a náklady představují podíl regulérních mailů, které jsou odmítnuty.

### Restrictions

Jak název napovídá, restrictions jsou omezení různého typu a v různé fázi průběhu odesílání nebo příjmu pošty dle SMTP protokolu. Znalost tohoto protokolu, alespoň rámcově, je pro pochopení vítaná až nutná. Pokud SMTP protokol neznáte, přečtěte si první díl seriálu o poštovním serveru nebo samotné RFC 5321. Restrictions umožňují omezit množství propuštěného spamu využitím mezer v softwaru spammerů, resp. faktu, že tento software často nerespektuje příslušná RFC nebo jejich doporučení. Problémem je, jak je již naznačeno výše, že tato omezení mohou odmítnout i regulérní e-mail přicházející z ne zcela vhodně nakonfigurovaného poštovního serveru.

Následují proměnné definující jednotlivá omezení:

- smtpd_client_restrictions – omezení SMTP klienta
- smtpd_helo_restrictions – omezení vztažené k HELO/HELO příkazu
- smtpd_sender_restrictions – omezení dle původce e-mailu (příkaz MAIL FROM: v rámci SMTP relace)
- smtpd_recipient_restrictions – jediná povinná položka, omezení vztažená k RCPT TO:
- smtpd_data_restrictions – omezení vztažená k příkazu DATA
- smtpd_end_of_data_restrictions – omezení vztažená k příkazu ukončujícímu přenos dat
- smtpd_etrn_restrictions – omezení příkazu ETRN

Tyto proměnné obsahují seznam pravidel (oddělených čárkou). Některá pravidla jsou společná (např. check_policy_service, reject či permit), jiná jsou specifická pro daný typ omezení. Podrobný popis jednotlivých omezení a všech možných hodnot naleznete v dokumentaci Postfixu, tj. buď v online podobě, nebo pomocí příkazu:

```
man 5 postconf
```

Připomínám, že tato omezení se definují v /etc/postfix/main.cf. Aktuálně nastavená omezení si můžete nechat vypsat příkazem:

```
postconf | grep restrictions
```

Aby to nebylo tak jednoduché, existují ještě další proměnné vztažené k daným omezením. Kupříkladu, smtpd_helo_required řídí, zda je příkaz HELO (či EHLO) vyžadován, nebo jej klient nemusí posílat. Pravidla umístěná v proměnné smtpd_helo_restrictions pak definují, jakým podmínkám musí příkaz HELO vyhovět (byl-li v průběhu SMTP relace zadán). Těchto proměnných je celá řada (viz dokumentace).

Ještě než se dostanu k samotným příkladům, zmíním jednu důležitou související proměnnou, a sice smtpd_delay_reject, kterou doporučuji nastavit na „yes":

```
smtpd_delay_reject = yes
```

Tato volba zajistí, že v případě nevyhovění některému z pravidel nedojde k okamžitému odmítnutí, ale vyčká se s odmítnutím až na fázi po zadání RCPT TO:. To má jednak výhodu v kompatibilitě s některými špatně napsanými klienty, ale mnohem podstatnější je skutečnost, že díky tomu bude moci server v případě odmítnutí zaznamenat odesílatele i příjemce zprávy (což se pak dozvíte z logu).

#### *smtpd_helo_restrictions*

Příklad pro smtpd_helo_restrictions, omezení vztažená k příkazu HELO/EHLO následuje:

```
smtpd_helo_required = yes
smtpd_helo_restrictions =
    permit_mynetworks,
    reject_invalid_helo_hostname,
    reject_non_fqdn_helo_hostname,
    permit
```

Seznam pravidel se prochází směrem od prvního k poslednímu, jak je obvyklé. První pravidlo, permit_mynetworks, propustí klienta z rozsahu definovaného v proměnné mynetworks, aniž by musel vyhovět kterémukoliv dalšímu pravidlu.

Pravidlo reject_invalid_helo_hostname odmítne klienta, pokud je jím použitá syntaxe hostname příkazu HELO/EHLO neplatná.

Pravidlo reject_non_fqdn_helo_hostname pak dále požaduje, aby klient uvedl jako hostname plně kvalifikované doménové jméno, jak požaduje RFC.

Nezachytí-li se klient u některého z předchozích pravidel, dostane se na konec seznamu, kde je pravidlo permit, které ho pustí dál.

Rozhodnete-li se nasadit tato omezení, bývá dobré také požadovat, aby klient příkaz HELO/EHLO uváděl (to je realizováno na prvním řádku příkladu, pomocí proměnné smtpd_helo_required).

#### *smtpd_sender_restrictions*

Tato omezení se vztahují k příkazu MAIL FROM: v rámci SMTP relace. Pozor – pole From: může následovat i v hlavičce mailu, kde už jeho obsah nehraje (v rámci tohoto omezení) roli.

```
smtpd_sender_restrictions =
    permit_mynetworks,
    reject_non_fqdn_sender,
    reject_unknown_sender_domain,
    permit
```

Struktura je podobná jako v příkladě výše. Pravidlo reject_non_fqdn_sender odmítne odesílatele, který nemá plně kvalifikované doménové jméno, jak je požadováno v RFC. Pravidlo reject_unknown_sender_domain pak odmítne klienta, který v doménové části e-mailu použije doménu, která nemá definovaný A ani MX záznam nebo která má nekorektní MX záznam. V případě dočasné chyby DNS vrací chybový kód 450 (čímž instruuje klienta, aby zkusil doručit mail později).

#### *smtpd_recipient_restrictions*

Tato omezení se vztahují k příkazu RCPT TO: v rámci SMTP relace. Jako jediné ze všech omezení je povinné a jeho výchozí hodnota je tato:

```
smtpd_recipient_restrictions = permit_mynetworks, reject_unauth_destination
```

Klíčové je zde pravidlo reject_unauth_destination, které odmítá poštu pro všechny příjemce s výjimkou těch, pro které váš poštovní server představuje konečnou. Toto pravidlo tedy zajišťuje, že server neslouží jako open relay, který je hojně využíván spammery k přeposílání spamu. Vždy se ujistěte, že toto pravidlo ve své konfiguraci máte.

Příklad komplexnějšího nastavení by mohl vypadat takto:

```
smtpd_recipient_restrictions =
    reject_unauth_pipelining,
    reject_non_fqdn_recipient,
    reject_unknown_recipient_domain,
    permit_mynetworks,
    reject_unauth_destination,
    check_sender_access
        hash:/etc/postfix/sender_access,
    reject_rbl_client my_blacklist.example.org,
    permit
```

Vezměme to pěkně popořadě. Pravidlo reject_unauth_pipelining odmítne klienta, který používá techniku zvanou pipelining, umožňující urychlit doručování většího množství e-mailů použitím více SMTP příkazů najednou. Tato technika vyžaduje, aby si klienti nejprve ověřili, je-li podporována. Spammeři často rovnou posílají řadu příkazů bez toho, aby prováděli ověření. A právě takového klienta Postfix v tomto případě odmítne.

Pravidlo reject_non_fqdn_recipient je díky předchozím příkladům už jasné (odmítne příjemce, který nepoužívá plně kvalifikované doménové jméno), stejně jako reject_unknown_recipient_domain (odmítne příjemce s neexistující doménou).
Pravidla permit_mynetworks i reject_unauth_destination byla popsána výše.
Zbývají tedy dvě, check_sender_access a reject_rbl_client. První z nich prověří místní databázi povolených nebo zakázaných odesílatelů, zatímco reject_rbl_client umožňuje napojení na některý z DNS blacklistů. Více viz dále.
check_sender_access
Toto pravidlo vyžaduje jako parametr soubor s dodatečnými pravidly (v příkladu výše/etc/postfix/sender_access). Tento soubor může vypadat třeba takto:

```
hodny_odesilatel.tld    OK
spammer.tld             REJECT
zly.spammer.tld         REJECT You have been blacklisted.
zly@spammer.tld         REJECT
zloun@                  REJECT
spammer@                REJECT
```

Syntaxe by z tohoto příkladu měla být vcelku jasná. Lze specifikovat jak domény, tak jména (část před zavináčem). Tímto způsobem lze realizovat místní blacklist i whitelist, propuštění e-mailu zajistí OK, odmítnutí pak REJECT, přičemž REJECT můžete následovat zprávou, která se pak pošle jako součást chybové hlášky. Můžete tak například těm, kteří se na váš blacklist dostali, poskytnout nějakou metodu, jak se z něj nechat vyjmout.
Změny v souboru vyžadují vygenerování databáze, podobně jako v případě aliasů:

```
postmap /etc/postfix/sender_access
```

Existuje podobné pravidlo check_recipient_access, které umožňuje realizovat totéž na úrovni příjemců a pro každého příjemce (nebo skupinu příjemců) umí použít jinou politiku přístupu. Příklad využití naleznetezde.
reject_rbl_client a blacklisty
Blacklisty byly představeny v tomto seriálu již dříve, zopakuji tedy jen to nejdůležitější. Blacklist je služba, která sdružuje na základě nějaké politiky IP adresy nebo IP rozsahy a umožňuje poštovním serverům prostřednictvím specifických DNS dotazů zkontrolovat, zda se IP adresa SMTP klienta v této databázi vyskytuje, či nikoliv.
Blacklisty na této úrovni v Postfixu je možné využít pouze k odmítání pošty, což nebývá vždy ten nejlepší nápad. Řada správců poštovních serverů raději blacklisty zařazuje až do fáze kontroly antispamem (např. spamassassin), kde je příslušným e-mailům pouze zvýšeno skóre, ale nejsou natvrdo odmítány.
Úmyslně v příkladu výše neuvádím žádný známý blacklist – jednak nechci dělat reklamu zadarmo, a pak, místo slepého cut-and-paste, doporučuji blacklisty vybírat ručně, a to velmi pečlivě. Je nepříjemné, pokud zvolíte nějaký, který po správcích serverů, kteří se na blacklist třeba dostali omylem nebo ne vlastní chybou, vyžaduje poplatek za vyřazení z databáze. Stejně tak pečlivě prověřte, jaké IP adresy jsou na blacklist zařazovány, zda patří počítačům odesílajícím spam, nebo pouze majitelům IP adres specifických typů poskytovatelů (např. běžným ISP) – nejeden linuxový/unixový nadšenec má domácí poštovní server, a použitím takového blacklistu byste jej natvrdo odřízli.
Některé blacklisty mohou mít tak nastavená pravidla, že je lze považovat v podstatě za vyděrače (zařazují na blacklist kdekoho a za vyjmutí si nechají zaplatit).
Stejně tak je třeba dát si pozor na pravidla použití daného blacklistu, některé lze použít třeba pouze pro nekomerční účely a pouze pro určitý objem pošty.
Použití blacklistů v Postfixu je snadné, postačí daný server uvést jako parametr pravidla reject_rbl_client:

```
smtpd_recipient_restrictions =
    ...
    reject_rbl_client my_blacklist.example.org,
    permit
```

Je to asi naprosto jasné, ale pro úplnost dodám, že v tomto příkladu by se DNS dotazy vázaly na servermy_blacklist.example.org.

## Závěr

Restrictions umožňují definovat komplexní pravidla přístupu, mnohem komplexnější, než co zde bylo probráno. Kupříkladu, řídit přístup lze i podle obsahu hlaviček e-mailu, které lze prověřovat prostřednictvím regulárních výrazů, a na jejich základě poštu odmítat nebo přijímat. A to je jen špička ledovce. Berte tedy tento článek pouze jako úvod do tématu s tím, že další možnosti řízení přístupu se dozvíte, pokud nahlédnete do dokumentace k Postfixu. Dokumentace k Postfixu je dobře strukturována, je dostatečně podrobná, jasná a v online podobě i vhodně prolinkovaná.

## Správa linuxového serveru: Uživatel a poštovní schránka

Po zprovoznění poštovního serveru určitě budete chtít mít možnost došlou poštu pohodlně vybírat, stejně jako poštu odesílat. V tomto dílu se dozvíte, jak toho docílit.

Pro odesílání pošty slouží protokol SMTP, ovšem Postfix nastavený podle dosavadních instrukcí, které jste nalezli v tomto seriálu, vám umožní odesílat poštu, pouze jste-li přihlášeni na server (leda byste si domynetworks přidali všechny externí počítače, ze kterých odesíláte poštu, což by však bylo krajně nepohodlné).

Za tímto účelem by bylo dobré implementovat SMTP autentifikaci, v rámci které se budete moci odkudkoliv k serveru přihlásit a jako přihlášený uživatel odeslat poštu. Jelikož posílat jméno a heslo přes internet v textové podobě je z bezpečnostního hlediska velkou chybou, měli byste také implementovat TLS.

Výběr pošty lze zrealizovat dvěma způsoby: prostřednictvím protokolů POP3 nebo IMAP4. Serverů zajišťujících POP3 a IMAP4 přístup k poštovním schránkám je více, já se zaměřím na Dovecot.

### *Webová rozhraní*

Kromě výše uvedených metod ještě přicházejí v úvahu webová rozhraní. Přímo v distribuci Debianu naleznete jednodušší Squirrelmail a komplexnější a „hezčí" Roundcube. Ta budou představena a zprovozněna v některém z dalších dílů.

### Odesílání pošty, SMTP AUTH a TLS

Autentifikaci neprovádí samotný Postfix, nýbrž démon saslauthd. Ten nainstalujete tímto způsobem:

```
aptitude install sasl2-bin
```

Následně upravte soubor s výchozími parametry démona saslauthd:

```
vim /etc/default/saslauthd
```

Řekněte startovacím skriptům, aby při startu systému rozběhli i saslauthd (ve výchozím stavu po instalaci saslauthd nenajede):

```
START=yes
```

Jelikož Postfix sám běží v chrootu, je nutné přemístit pojmenovanou rouru, přes kterou se Postfix a saslauthd propojují, což provedete úpravou proměnné OPTIONS na konci konfiguračního souboru:

```
OPTIONS="-c -m /var/spool/postfix/var/run/saslauthd"
```

Zbývá nastavit Postfix jako takový. Ve výchozím stavu by TLS měl již podporovat, pokud ne, doplňte domain.cf následující řádky:

```
smtpd_tls_cert_file=/cesta/k/ssl/certifikatu.pem
smtpd_tls_key_file=/cesta/k/soukromemu/klici.key
smtpd_use_tls=yes
smtpd_tls_auth_only = yes
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache
```

K šifrovanému spojení je pochopitelně třeba certifikát. Výchozí automaticky vygenerovaný certifikát je podepsaný sám sebou, což nemusí být z hlediska bezpečnosti úplně nejvhodnější (vaši uživatelé nemají jak si ověřit, zda je daný certifikát skutečně váš a ne podstrčený nějakou třetí osobou v rámci man-in-the-middle útoku). Máte-li certifikát k doméně poštovního serveru podepsaný nějakou známou certifikační autoritou, bývá dobré jej použít místo něj (v takovém případě upravte smtpd_tls_cert_file asmtpd_tls_key_file). Připomínám, že existují certifikační autority, kde můžete získat certifikát zdarma (např. CACert, StartSSL).

TLS jako takové zapíná volba smtpd_use_tls, přičemž vynucení autentifikace pouze přes TLS řídí volbasmtpd_tls_auth_only.

Kromě řádků výše je třeba zprovoznit i SASL autentifikaci. Přidejte tedy do main.cf následující řádky:

```
smtpd_sasl_auth_enable = yes
smtpd_sasl_security_options = noanonymous
broken_sasl_auth_clients = no
smtpd_sasl_authenticated_header = yes
```

Autentifikace pomocí SASL se zapíná proměnnou smtpd_sasl_auth_enable. Bývá dobré vypnout anonymní autentifikaci, tedy nastavit proměnnou smtpd_sasl_security_options na hodnotu noanonymous. Ne všichni klienti podporují aktuální verzi příkazu AUTH, jako např. MS Outlook Express 4 či Exchange 5. Chcete-li zajistit kompatibilitu vašeho serveru i s takovými klienty, nastavte broken_sasl_auth_clients na hodnotuyes. A konečně poslední proměnná (smtpd_sasl_authenticated_header) řídí, zda v hlavičkách e-mailu bude zanesena informace o tom, který autentifikovaný uživatel danou zprávu zaslal. Pro lepší názornost uvádím příklad takové hlavičky se znázorněním příslušného řádku:

```
Received: from localhost.localdomain (unknown [1.2.3.4])
(using TLSv1 with cipher DHE-RSA-AES128-SHA (128/128 bits))
(No client certificate requested)
(Authenticated sender: michal)
by server.example.net (Postfix) with ESMTPSA id 2A0D62970025
for <michal@server.example.net>; Thu, 31 May 2012 22:43:24 +0200 (CEST)
```

Výše uvedené nastavení řeklo Postfixu, aby podporoval TLS a SASL autentifikaci. Vy ale chcete uživatelům autentifikovaným přes SASL umožnit odesílat poštu. Za tímto účelem je ještě třeba upravit proměnnousmtpd_recipient_restrictions (viz minulý díl seriálu, kde byla tato problematika podrobně popsána). Na správné místo je třeba přidat permit_sasl_authenticated, obvykle za permit_mynetworks:

```
smtpd_recipient_restrictions =
reject_unauth_pipelining,
reject_non_fqdn_recipient,
reject_unknown_recipient_domain,
permit_mynetworks,
permit_sasl_authenticated,
reject_invalid_hostname,
reject_unauth_destination,
permit
```

Dodám, že výše uvedené nastavení smtpd_recipient_restrictions může být pro některé použití až příliš restriktivní, berte jej tedy pouze jako příklad a ne jako vzor, který je třeba následovat.

Zbývají dva poslední kroky. Nejprve je třeba vytvořit soubor /etc/postfix/sasl/smtpd.conf a zapsat do něj následující řádek:

```
pwcheck_method: saslauthd
```

Druhým krokem je přidání uživatele „postfix" do skupiny „sasl", čímž umožníte Postfixu komunikovat se saslauthd:

```
gpasswd -a postfix sasl
```

Poté restartujte saslauthd a Postfix:

```
service saslauthd restart
service postfix restart
```

Nyní můžete vyzkoušet nakonfigurovat poštovního klienta k odesílání pošty přes váš server. Nezapomeňte v jeho nastavení povolit TLS a SMTP autentifikaci.

### Čtení pošty, POP3, IMAP4 a Dovecot

Dovecot podporuje obě varianty obou protokolů pro přístup k poště, tedy POP3 a jeho SSL variantu POP3S, dále IMAP4 a jeho SSL variantu IMAPS. Umí samozřejmě podporovat i TLS, tedy šifrování na transportní vrstvě. Záleží jen na vás, které varianty zvolíte. Chcete-li zvolit všechny, upravte konfigurační soubor/etc/dovecot/dovecot.conf a upravte řádku s protokoly, takto:

```
protocols = imap imaps pop3 pop3s
```

Používáte-li SSL varianty protokolů nebo chcete-li použít TLS, je třeba opět provést patřičné nastavení:

```
ssl = yes
ssl_cert_file = /etc/ssl/certs/dovecot.pem
ssl_key_file = /etc/ssl/private/dovecot.pem
disable_plaintext_auth = yes
ssl = required
```

Většina voleb je zde asi jasná. disable_plaintext_auth zakáže plaintextovou autentifikaci a ssl = requiredzajistí, že bude SSL/TLS vyžadováno i pro neplaintextovou autentifikaci.
Aby Dovecot mohl přistupovat k vaší poště, je třeba mu říci, kde ji má hledat. k tomu slouží proměnnámail_location, která má poměrně komplexní syntax a která je dobře popsána přímo v konfiguračním souboru. Používáte-li výchozí nastavení, tj. schránky typu MBOX ve /var/mail/uzivatel, nastavtemail_location takto:

```
mail_location = mbox:~/mail:INBOX=/var/mail/%u
```

Pro to nejzákladnější nastavení by tyto volby měly stačit. Restartujte Dovecot a můžete začít testovat pomocí poštovního klienta:

```
service dovecot restart
```

Pro jednotlivé protokoly lze specifikovat, na kterých IP adresách a portech má démon naslouchat, viz následující příklad:

```
protocol imap {
  listen = 127.0.0.1:143
  ssl_listen = *:993
}
```

Hvězdička značí všechna rozhraní. Tento příklad by zajistil, že nešifrovaný IMAP je přístupný pouze lokálně, zatímco IMAPS je dostupný na všech síťových rozhraních serveru. Toto je samozřejmě pouze ilustrační příklad, pro zakázání nešifrované autentifikace tento postup není třeba – vhodnější je zakázat plaintextovou autentifikaci (viz výše).
Podobným způsobem je také možné každé službě přiřadit jiný SSL certifikát – stačí použít příslušné volby (ssl_cert_file a ssl_key_file) taktéž uvnitř sekce protocol. Možnosti konfigurace Dovecotu jsou velmi bohaté, proto doporučuji se podívat do manuálových stránek nebo na wiki Dovecotu.

## Správa linuxového serveru: Webová rozhraní k poště (Squirrelmail a Roundcube)

Tento díl je věnován možnostem a konfiguraci webových rozhraní pro přístup k poštovní schránce, konkrétně pak Squirrelmailu a Roundcube.

V minulém díle byly představeny možnosti klientského přístupu k poště. V dnešní době jsou k dispozici dvě možnosti. Jednou z nich je přístup přes poštovního klienta pomocí protokolů IMAP či POP3 a SMTP. Druhou je pak přístup přes webovou aplikaci. Běžní uživatelé jsou v dnešní době spíše zvyklí používat webové rozhraní než poštovního klienta (nemusí se nic nastavovat a k poště lze přistupovat z libovolného počítače).

V tomto díle budou představena dvě webová rozhraní, jednodušší Squirrelmail a komplexní Roundcube. Krátký popis i snímky obrazovky jsou u každého z nich.

### *Bezpečnostní doporučení: používejte HTTPS*

Ať už si vyberete jakékoliv webové rozhraní, nelze než důrazně doporučit celé rozhraní provozovat pouze přes HTTPS, aby nedocházelo k odesílání jména a hesla přes nešifrované HTTP spojení.

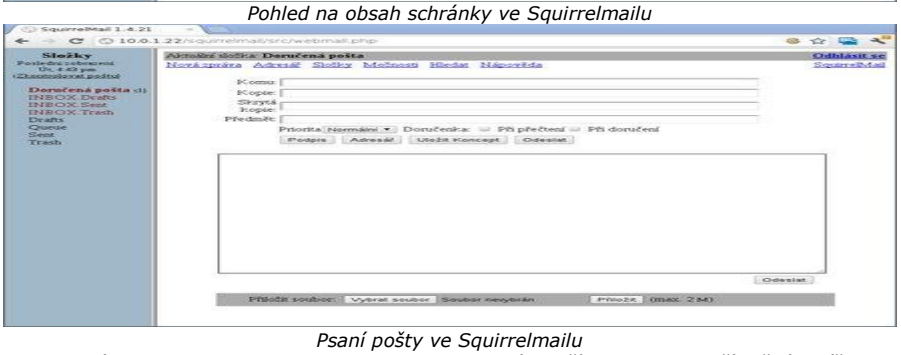
### *Bez IMAP serveru to nejde*

Obě webová rozhraní využívají pro přístup k poštovní schránce protokol IMAP, je tedy nutné mít IMAP server zprovozněný (návod, jak na to, naleznete v minulém díle). Nechcete-li nabízet přístup přes IMAP celému světu, nakonfigurujte IMAP server tak, aby naslouchal jen na localhostu. Pokud neomezujete odchozí SMTP pocházející přímo ze serveru (localhost), není třeba žádná speciální konfigurace pro odesílání pošty.

### Squirrelmail

Squirrelmail je jednoduché webové rozhraní napsané v PHP bez pokročilých funkcí. Není to „Web 2.0" aplikace, ale svou úlohu umí zastat velmi dobře. Podporuje dokonce i adresář s kontakty, jehož data jsou uchovávána v textových souborech. Squirrelmail si můžete prohlédnout na obrázcích níže:

*Pohled na obsah schránky ve Squirrelmailu*



*Psaní pošty ve Squirrelmailu*

Samotná instalace Squirrelmailu je velmi jednoduchá, stačí nainstalovat příslušný balíček:

```
aptitude install squirrelmail
```

Konfigurace vyžaduje ještě několik ručních zásahů. Asi je jasné, že aplikace zajišťující webové rozhraní k poště vyžaduje vhodné nastavení webového serveru. Konfigurační soubor nastavující službu na Apachi je k dispozici v /etc/squirrelmail/apache.conf. Je to vzorový soubor, takže si jej přizpůsobte. Pokud hostujete více virtuálních webů, je více než vhodné integrovat potřebné direktivy do příslušného virtuálního webu (ideálně do virtuálního webu chráněného SSL, viz výše). Pro to nejzákladnější nastavení, kdy je vytvořen alias /squirrelmail pro stávající konfiguraci, postačí vytvořit symbolický odkaz do /etc/apache2/conf.d, takto:

```
ln -s /etc/squirrelmail/apache.conf /etc/apache2/conf.d/squirrelmail
```

A restartovat webový server:

```
service apache2 restart
```

Nyní by již měl Squirrelmail fungovat. Pokud vám vadí angličtina a chtěli byste svým uživatelům poskytnout (převážně) české prostředí, upravte konfigurační soubor /etc/squirrelmail/config_local.php a přidejte do něj následující řádky se specifikací výchozího jazyka a znakové sady:

```
$squirrelmail_default_language = 'cs_CZ';
$default_charset       = 'utf-8';
```

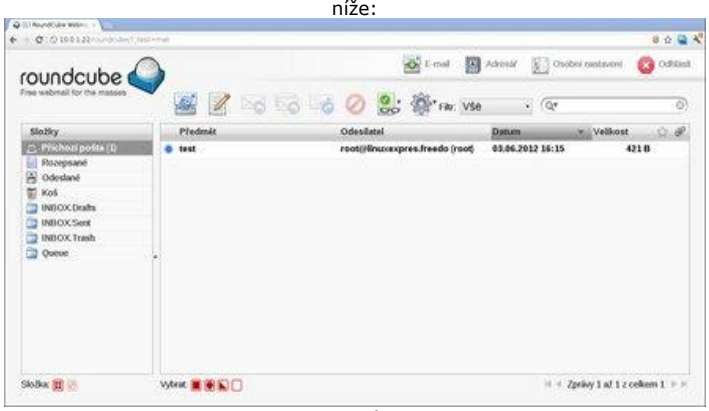
Změna by měla být okamžitá, není třeba restartovat webový server. Kromě běžných funkcí nabízí Squirrelmail i několik pluginů, které jeho funkcionalitu rozšiřují. Jejich seznam získáte snadno, vypsáním balíčků obsahujících „squirrelmail" v názvu:

```
aptitude search squirrelmail
```

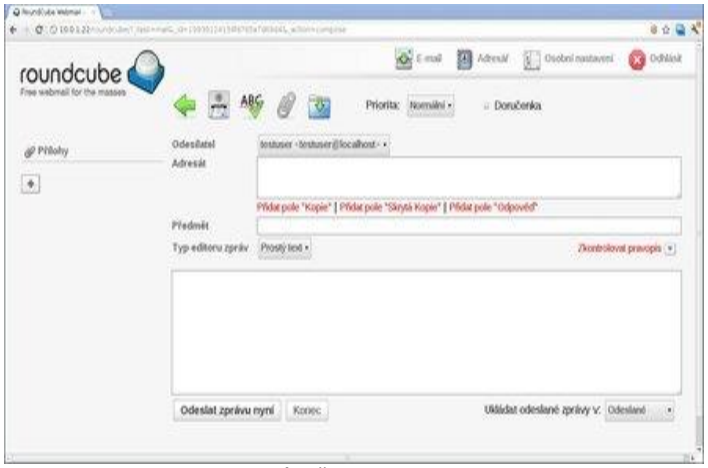
Z těch zásadnějších doporučuji alespoň squirrelmail-quicksave, který nabízí, jak je z názvu patrné, automatické ukládání e-mailů po určité době.

### Roundcube

Roundcube je oproti Squirrelmailu komplexnější, nabízí více funkcí a „Web 2.0" rozhraní (tzn. AJAX), ovšem je také náročnější na zdroje – na rozdíl od Squirrelmailu, který si vystačí s textovými soubory, vyžaduje databázi (MySQL, PostgreSQL nebo SQLite). Názor si můžete udělat z obrázků níže:




*Pohled na obsah schránky v Roundcube*




*Psaní pošty v Roundcube*

Nemáte-li dosud nainstalovaný databázový server a nechcete-li používat SQLite, můžete nainstalovat třeba MySQL:

```
aptitude install mysql-client mysql-server
```

V takovém případě nezapomeňte nainstalovat také PHP knihovnu, která tvoří most mezi PHP aplikacemi a databází MySQL (php5-mysql), a také MySQL knihovnu, kterou používá samotný Roundcube (roundcube-mysql):

```
aptitude install php5-mysql roundcube-mysql
```

Máte-li nainstalováno všechno potřebné, co se týče databáze, samotnou instalaci Roundcube provedete stejně jako v případě Squirrelmailu:

```
aptitude install roundcube
```

Během instalace budete provedeni vytvořením databáze pro Roundcube. Pokud byste cokoliv opomněli, můžete konfiguraci provést kdykoliv znovu, a to pomocí příkazu:

```
dpkg-reconfigure roundcube-core
```

Nyní je třeba provést základní nastavení. Jako první je třeba vyřešit propojení s webovým serverem, které je v případě Roundcube automatické (resp. je vytvořen symbolický link v /etc/apache2/conf.d odkazující na/etc/roundcube/apache.conf. Tento odkaz buď vymažte a obsah tohoto souboru asimilujte do svého nastavení Apache, nebo tento soubor upravte a odkomentujte dva řádky s definicí aliasů:

```
Alias /roundcube/program/js/tiny_mce/ /usr/share/tinymce/www/
Alias /roundcube /var/lib/roundcube
```

Následně upravte soubor /etc/roundcube/main.inc.php, ve kterém je třeba definovat povolené servery, ke kterým se instance Roundcube na vašem serveru bude smět připojovat. Chcete-li používat Roundcube pro poštu pouze na vašem serveru, tedy na localhostu, proveďte úpravu proměnné$rcmail_config['default_host'] takto:

```
$rcmail_config['default_host'] = 'localhost';
```

Chcete-li umožnit uživatelům připojovat se i k jiným IMAP serverům z vašeho serveru, můžete využít následujícího vzoru:

```
$rcmail_config['default_host'] = array('localhost', 'ssl://mail.example.org:993');
```

Pokud tuto volbu ponecháte prázdnou (což je výchozí nastavení), uživatelé se budou moci hlásit k libovolnému serveru (což se dá považovat za bezpečnostní riziko). Poté zbývá už jen restart Apache:

```
service apache2 restart
```

## Správa linuxového serveru: Antivirus a antispam

Na většině linuxových poštovních serverů běží dva základní nástroje pro boj se spamem, a sice SpamAssassin v roli antispamu a ClamAV v roli antivirového řešení. V tomto díle se dozvíte, jak je nainstalovat a nastavit i na vašem poštovním serveru.

Mnohé předešlé díly se věnovaly řešením, které by měly pomoci omezit množství spamu, které váš server propustí. Za jedno z nejúčinnějších řešení je považován greylisting, ale není to jediná technika, která může pomoci.

Dosud probírané techniky mají však jeden společný faktor – omezují spam na základě způsobu odesílání, tzn. kdo e-mail poslal, jak se jeho SMTP server choval a jak byl nastaven. To úspěšně odfiltruje řadu automatizovaných nástrojů pro posílání spamu (a bohužel i některé špatně nastavené poštovní servery), ale spam pocházející z korektně pracujících poštovních serverů, které (zatím) nejsou na žádných blacklistech, uvedené metody nezadrží. Zde je třeba nasadit poslední obrannou linii, kontrolu obsahu.

### Amavis, SpamAssassin, ClamAV

Kontrolu obsahu zajišťuje **SpamAssassin**, nejrozšířenější antispamové řešení (nejen) na linuxových serverech. Je napsaný v Perlu a pracuje tak, že na základě sady pravidel prověřuje průchozí poštu. V případě, že konkrétnímu pravidlu e-mail vyhoví, upraví se jeho celkové skóre o počet bodů, který náleží danému pravidlu. Kupříkladu, obsahuje-li tělo e-mailu slovo „viagra“, zvýší se skóre o 2,7 bodu. Některá pravidla mohou e-mailu skóre i snížit. Výsledné skóre se pak porovná s prahovou hodnotou (výchozí je 5,0 bodů) a v případě, že e-mail hodnotu dosáhne nebo překročí, je označen jako spam.

Skóre u jednotlivých pravidel nejsou navrhována náhodně, ale měla by při prahové hodnotě 5,0 bodů dosahovat jednoho falešného pozitiva (korektního e-mailu označeného jako spam) v dvou a půl tisících regulérních e-mailech (tzv. hamech).

Prahovou hodnotu i skóre jednotlivých pravidel si můžete upravit. Dokonce si můžete tvořit i vlastní pravidla. Podstatnou informací je i to, že sada pravidel SpamAssassinu se podobně jako v případě antivirových databází časem mění a aktualizuje. Pokud chcete, můžete pomocí cronu aktualizovat nejenom antivirovou databázi, ale i pravidla pro SpamAssassin. Více v návodu níže.

Pokud uživatelé vašeho serveru nepoužívají operační systém Microsoft Windows, v podstatě antivirus nepotřebujete. Máte-li však takové uživatele, je vhodné je chránit pomocí antivirového řešení, které bude hledat viry v poště a likvidovat je. Základním antivirovým nástrojem v GNU/Linuxu je **ClamAV**.

Je třeba zmínit, že testování obsahu e-mailů pomocí SpamAssassinu a ClamAV je náročné na výkon serveru (nejenom na CPU, ale i na paměť), což je i důvod, proč se správci poštovních serverů tolik věnují ostatním metodám obrany proti spamu. Čím více toho odfiltrují restrikce, greylisting apod., tím méně toho zbude na antispam, potažmo antivirus.

A k čemu že je **Amavis**? Amavis je spolehlivý nástroj, opět napsaný v Perlu, který stojí mezi vaším MTA a filtry obsahu v podobě SpamAssassinu a ClamAV.

### Instalace

Instalaci všech tří nástrojů provedete jednoduše, přes správce balíčků, v případě Debianu takto:

```
aptitude install amavisd-new spamassassin clamav-daemon
```

Schopnosti detekce antivirů můžete posílit, pokud nainstalujete nástroje pro dekompresi souborů (díky tomu bude ClamAV schopen prověřovat i obsahy archívů). Pozor! Některé z těchto nástrojů nepatří mezi svobodný software, tudíž důrazně doporučuji projít jejich licenční ujednání před tím, než je nasadíte na server. Také budete v tomto případě potřebovat contrib a non-free repozitáře Debianu.

```
aptitude install arj bzip2 cabextract cpio file gzip lha nomarch pax unrar unzip zip
```

### Nastavení ClamAV

Nejprve přidejte uživatele *clamav* do skupiny *amavis*, aby měl ClamAV přístup k souborům pro antivirový sken:

```
gpasswd -a clamav amavis
```

Následně ClamAV démona restartujte:

```
service clamav-daemon restart
```

### Nastavení SpamAssassinu

Tím je nastavení ClamAV hotové. Zbývá nastavit SpamAssassin. Upravte soubor/etc/default/spamassassin a nastavte proměnnou ENABLED na jedničku:

```
ENABLED=1
```

Chcete-li, aby se pravidla SpamAssassinu aktualizovala každý den pomocí cronu, ve stejném souboru nastavte proměnnou CRON také na jedničku:

```
CRON=1
```

Jak už bylo řečeno výše, činnost SpamAssassinu je náročná na procesor, tudíž nemusí být úplně od věci nastavit SpamAssassinu nízkou prioritu, aby se přednostně vyřizovaly aktuální požadavky ostatních démonů. Nejnižší priorita znamená nejvyšší „niceness“ hodnotu, tedy 19, standard je 0 a nejvyšší priorita (nejnižší „niceness“) je –20. Rozumně nízká priorita (15) je v konfiguračním souboru připravená, stačí ji pouze odkomentovat (popřípadě upravit):

```
NICE="--nicelevel 15"
```

Poté nastartujte SpamAssassin:

```
service spamassassin start
```

### Nastavení Amavisu

Ve výchozím nastavení Amavis neprovádí ani antispamovou, ani antivirovou kontrolu. Je tedy třeba mu říci, aby je prováděl, a to v souboru /etc/amavis/conf.d/15-content_filter_mode, kde je třeba odkomentovat následující dva bloky:

```
@bypass_virus_checks_maps = (
\%bypass_virus_checks, \@bypass_virus_checks_acl, \$bypass_virus_checks_re);

@bypass_spam_checks_maps = (
\%bypass_spam_checks, \@bypass_spam_checks_acl, \$bypass_spam_checks_re);
```

Když se podíváte do /etc/amavis/conf.d, uvidíte jednotlivé soubory obsahující nastavení Amavisu. Projděte si minimálně soubor /etc/amavis/conf.d/20-debian_default, kde jsou výchozí nastavení od tvůrců Debianu. V případě Debianu Squeeze tato nastavení nejsou optimální a je třeba je upravit (viz dále). Tento soubor však neměňte, pouze příslušné proměnné nadefinujte dle svých požadavků v souboru/etc/amavis/conf.d/50-user, který je pro lokální úpravy určen.

Do tohoto souboru můžete umístit tyto hodnoty:

```
@local_domains_acl = ( ".$mydomain" );

$sa_tag_level_deflt  = -999;
$sa_tag2_level_deflt = 5;
$sa_kill_level_deflt = 12;

$final_spam_destiny       = D_DISCARD;
```

První řádek definuje domény, pro které má Amavis filtrovat poštu – domény, které zde nebudou uvedeny, nebude Amavis filtrovat (pokud vám tedy Amavis nefiltruje poštu, prověřte, zda jsou zde uvedeny všechny vaše domény). Máte-li jich více, můžete je oddělit čárkou:

```
@local_domains_acl = ( ".$mydomain", "example.com", "example.org" );
```

Proměnná @sa_tag_level_deflt udává, od jakého skóre se do e-mailu budou přidávat hlavičky X-Spam-*. Hodnota –999 zafunguje pro všechny e-maily. Pro úvodní ladění a testování je to více než vhodné, jelikož se sem vypisují i hodnoty jednotlivých testů, což vám pomůže identifikovat, proč byl e-mail označen jako spam. Proměnná @sa_tag2_level_deflt udává, od jakého skóre má být daný e-mail označen jako spam. A konečně proměnná $sa_kill_level_deflt udává, od jaké úrovně se má se spamem provést akce definovaná ve $final_spam_destiny.

Výchozí nastavení Debianu Squeeze obsahuje v této proměnné D_BOUNCE, což je pro mne nepochopitelná hodnota – tato hodnota totiž způsobí, že se e-mail s vysokým skóre odmítne a vygeneruje se zpráva o nepřijetí, která se pošle odesílateli. Jak ale sami víte, odesílatel se dá snadno zfalšovat a u spamu bývá falšován téměř vždy. Pokud byste tuto hodnotu ponechali ve výchozím stavu, váš server by na spamy „odpovídal“, což by vyústilo v tzv. backscatter, tedy odesílání e-mailů o nepřijetí nevinným osobám. Doporučuji tedy nastavit buď na D_DISCARD (zahodit, popřípadě dle konfigurace poslat do karantény) neboD_PASS (propustit).

Následně restartujte Amavis:

```
service amavis restart
```

## Nastavení Postfixu

Nejprve vložte do /etc/postfix/main.cf řádku:

```
content_filter = smtp-amavis:[127.0.0.1]:10024
```

Nyní je třeba definovat příslušnou službu v souboru /etc/postfix/master.cf:

```
smtp-amavis unix    -       -       n       -       2       smtp
    -o smtp_data_done_timeout=1200
    -o smtp_send_xforward_command=yes
    -o disable_dns_lookups=yes
    -o max_use=20

127.0.0.1:10025 inet n  -       n       -       -       smtpd
    -o content_filter=
    -o smtpd_delay_reject=no
    -o smtpd_client_restrictions=permit_mynetworks,reject
    -o smtpd_helo_restrictions=
    -o smtpd_sender_restrictions=
    -o smtpd_recipient_restrictions=permit_mynetworks,reject
    -o smtpd_data_restrictions=reject_unauth_pipelining
    -o smtpd_end_of_data_restrictions=
    -o smtpd_restriction_classes=
    -o mynetworks=127.0.0.0/8
    -o smtpd_error_sleep_time=0
    -o smtpd_soft_error_limit=1001
    -o smtpd_hard_error_limit=1000
    -o smtpd_client_connection_count_limit=0
    -o smtpd_client_connection_rate_limit=0
    -o receive_override_options=no_header_body_checks,no_unknown_recipient_checks,no_milters
    -o local_header_rewrite_clients=
```

Amavis jako takový naslouchá na portu 10024. Postfix musí Amavisu předat poštu pro filtrování na tento port, ale zároveň je třeba, aby mu Amavis poštu předal po filtrování zpět, což se řeší výše uvedenou definicí služby na portu 10025. Na tomto portu pak Postfix bude od Amavisu přijímat výsledek filtrování.

Na závěr je třeba Postfixu oznámit změnu konfigurace:

```
service postfix reload
```

## Otestování funkčnosti

Pro základní otestování postačí, když pošlete na server e-mail, do kterého umístíte heslo „viagra" – to poměrně spolehlivě zvýší skóre na nenulovou hodnotu. Můžete také využít GTUBE řetězec, který ve výchozím nastavení přiřadí vašemu mailu skóre 999. Tento řetězec má tuto podobu:

```
XJS*C4JDBQADN1.NSBN3*2IDNEN*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL*C.34X
```

## Slovo závěrem

Amavis je komplexní nástroj, který toho umí poměrně hodně. Pokud jej na svůj server nasadíte, doporučuji vám projít si jeho dokumentaci (můžete začít u odkazů v závěru článku), zejména pak hodnoty v souboru/etc/amavis/conf.d/20-debian_defaults.

## Správa linuxového serveru: Poštovní server a více domén

Ve chvíli, kdy máte k dispozici více domén, vám už patrně ta nejjednodušší konfigurace poštovního serveru vázaná na unixové účty nebude stačit a začnete se poohlížet po možnostech tvorby virtuálních účtů pro jednotlivé domény. V tomto díle představíme jedno z takových řešení, postavené na webové aplikaci Postfixadmin.

Dosavadní konfigurace poštovního serveru představená v tomto seriálu počítala výhradně s vazbou poštovní schránky na unixové účty. Toto řešení je sice velmi jednoduché na implementaci, ale má několik nevýhod. V první řadě je to nutnost vytvořit unixový účet všem majitelům poštovních schránek. Pokud byste takovému uživateli nechtěli poskytnout přístup k shellu nebo jiným službám vázaným na unixové účty (což je z bezpečnostních důvodů více než rozumné), museli byste to nějakým způsobem ošetřit všude tam, kde se unixové účty používají.

Obsluhuje-li váš server více domén, mohli jste narazit na problém se stejnými jmény na různých doménách. Představte si, že máte unixového uživatele michal a domény example.com a example.net, přičemž poštovní schránky michal@example.com a michal@example.net mají mít jiné vlastníky. Ve výchozí konfiguraci Postfixu by pošta z obou adres směřovala do schránky uživatele michal. Jedno z řešení tohoto problému, spočívající ve vytvoření virtuálních aliasů, které ukazují na různé unixové účty (tedy např. michal@example.com by ukazoval na michal1 a michal@example.net by ukazoval na účet michal2), bylo představeno v tomto dílu. Takové řešení lze použít, dokud nemáte větší množství uživatelů nebo domén.

Jinou možností řešení tohoto problému jsou plně virtuální účty, tzn. zcela izolované účty pro poštu, s vlastním heslem (nezávislým na unixovém) pro každou schránku. Existuje mnoho způsobů, jak takové řešení implementovat. Často se jako úložiště nepoužívají textové soubory, nýbrž databáze (což bude i případ tohoto návodu). Postavíte-li si vlastní řešení, budete si muset ošetřit zakládání, rušení a nastavování účtů. V tomto díle bude uvedeno připravené řešení s využitím Postfixadminu, který správu účtů a domén usnadňuje.

### Postfixadmin

Postfixadmin je webová aplikace napsaná v PHP, určená pro správu virtuálních poštovních účtů. Má bohaté možnosti konfigurace a poměrně jednoduché rozhraní. Postfixadmin není zázračný nástroj, který vám zpřístupní úplně všechny možnosti nastavení Postfixu v tomto ohledu, ale umožní vám celkem snadnou správu virtuálních účtů.

### Virtuální účty, Postfix a Postfixadmin

Možnosti Postfixu a jeho nastavení jsou nepřeberné, což by mělo jít částečně poznat i na tomto seriálu, jelikož poštovnímu serveru se věnovalo předchozích deset dílů, což je mnohem více, než bylo věnováno jakémukoliv jinému tématu, a to byly probírány stále spíše jen základy. Řešení popsané v tomto dílu lze také považovat za základní řešení, od kterého se můžete odrazit, přičemž pokročilejší funkce typu kvót či automatické odpovědi v případě nepřítomnosti jsou ponechány na vašem dalším zkoumání Postfixadminu a Postfixu.

Zatímco nastavit Postfixadmin je relativně jednoduché, ke zprovoznění celého řešení bude třeba upravit konfiguraci Postfixu a IMAP/POP3 démona, které nejsou úplně triviální a není těžké v nich udělat chybu. Proto tentokrát více než kdy jindy důrazně nedoporučuji provádět podobné úpravy na produkčních serverech. Doporučuji si toto řešení vyzkoušet někde nanečisto před nasazením.

### Instalace

Předpokladem pro instalaci Postfixadminu jsou následující komponenty:

- webový server (optimálně Apache nebo Lighttpd – tyto dva servery Postfixadmin automaticky nastaví při instalaci)
- databázový server (MySQL nebo PostgreSQL, návod bude počítat s MySQL)
- PHP interpret (balíček `php5` s příslušným konektorem k databázovému serveru, tj. `php5-mysql` nebo`php5-pgsql`)
- Postfix s podporou vámi zvolené databáze (balíček `postfix-mysql` nebo `postfix-pgsql`)
- IMAP/POP3 server (Dovecot či Courier, návod bude počítat s Dovecotem)

Nemáte-li výše uvedené komponenty nainstalované, učiňte tak před instalací Postfixadminu.

Podstatnou informací je, že Postfixadmin není součástí repozitářů Debianu Squeeze. K dispozici je až pro Debian 7.0 s označením Wheezy.

Používáte-li Debian Squeeze, stáhněte si deb balíček ze stránek projektua nainstalujte jej pomocí nástroje dpkg:

```
dpkg -i balicek.deb
```

Používáte-li Debian Wheezy, mělo by stačit použít Aptitude:

```
aptitude install postfixadmin
```

Pozor – budete-li využívat Postfixadmin na Debianu Squeeze, budete muset řešit aktualizace Postfixadminu ručně. Naopak, používáte-li Debian Wheezy, berte na vědomí, že návod je otestován pouze v Debianu Squeeze. Zní to komplikovaně, ale není :)

Během instalace vás Debconf vyzve, abyste vybrali webový server pro automatickou konfiguraci a abyste vybrali databázový server. Následující text bude předpokládat MySQL jako databázový server. Chcete-li používat PostgreSQL, nahlédněte do dokumentace (některé databázové dotazy se budou lišit, nestačí tedy jen přepsat mysql za pgsql). Návod pro PostgreSQL naleznete i v odkazech pod článkem.

### Integrace s webovým serverem

Po instalaci by měl být váš webový server pro Postfixadmin nastavený, tedy za předpokladu, že jste použili některý z výše uvedených (Apache nebo Lighttpd). Používáte-li Apache, naleznete jeho konfigurační soubor v /etc/postfixadmin/apache.conf (v případě Lighttpd je to /etc/postfixadmin/lighttpd.conf). Vzhledem k povaze aplikace lze důrazně doporučit omezení přístupu k ní jednak na nějaký SSL virtuální web, aby hesla neputovala k serveru nešifrovaná, a jednak libovolným dalším způsobem dle vlastního uvážení (omezení na IP adresy, HTTP autentifikace atd.).

Úprava je jednoduchá – nejprve odstraňte symbolický odkaz /etc/apache2/conf.d/postfixadmin, který vkládá daný alias do všech vašich virtuálních webů (virtual host). Následně si vyberte některý z vašich existujících virtuálních webů (ideálně takový, který je chráněn SSL), popřípadě si vytvořte nový a doplňte do něj vyznačenou řádku:

```
<VirtualHost 1.2.3.4:443>

ServerName www.example.cz
ServerAlias example.cz

Alias /postfixadmin /usr/share/postfixadmin

DocumentRoot /var/www/example.cz/www
<Directory "/var/www/example.cz/www">
Order allow,deny
Allow from all
</Directory>

SSLEngine On
SSLCertificateFile /etc/apache2/example-cz-certifikat.pem
SSLCACertificateFile /etc/apache2/ssl-ca.pem

</VirtualHost>
```

Volitelně pak doplňte omezení k přístupu na daný web. Alias si samozřejmě můžete upravit a místo /postfixadmin zvolit něco jiného, kratšího.

Dodám, že nastavení Apache, tvorbě virtuálních webů, práci s aliasy, SSL a omezení přístupu k webům na základě IP adres se věnoval *Úvod do konfigurace Apache* (1. díl a 2. díl). Budete-li pracovat se SSL, hodí se vám ještě díl o SNI.

### Nastavení Postfixadminu

Webový server byste v tuto chvíli měli mít nastavený a k Postfixadminu byste měli být schopni se dostat prostřednictvím aliasu `/postfixadmin`. Jako první byste měli spustit setup.php:

```
https://server.example.org/postfixadmin/setup.php
```

Nejprve budete požádáni o správcovské heslo. Jakmile jej zadáte, vygeneruje se vám řádka, kterou vložíte do /etc/postfixadmin/config.inc.php, resp. kde nahradíte existující proměnnou $CONF['setup_password']. Následně si vytvořte administrátorský účet (podmínkou je platný e-mail a správcovské heslo, které jste si nastavili dříve).

Nyní byste měli ještě jednou upravit /etc/postfixadmin/config.inc.php, projít si jej a přizpůsobit si jeho obsah. Přinejmenším si upravte následující proměnné:

- $CONF['admin_email'] e-mail, ze kterého se bude odesílat e-mail do nově založených poštovních schránek
- $CONF['default_aliases'] výchozí aliasy, které bude Postfixadmin vytvářet pro novou doménu, pokud jej o to požádáte
- $CONF['footer_link'] kam povede odkaz v patičce
- $CONF['footer_text'] jaký bude mít odkaz v patičce text

Dokumentace k Postfixadminu je po instalaci k dispozici v /usr/share/doc/postfixadmin.

**Logika ovládání Postfixadminu**

Hlavním uživatelem je správce, kterého jste si vytvořili výše. Správce může definovat další správce, kteří pak budou mít na starost nějaké domény (které, to můžete specifikovat). Zatímco hlavní správce může pořizovat zálohu databáze (karta **Backup**), tvořit domény a další správce, správci konkrétních domén tato privilegia nemají, jsou omezeni na konkrétní domény. V podobném duchu jsou omezené aliasy – alias vytvořený hlavním správcem nelze upravovat nebo mazat s právy správce domény. U poštovních schránek to však neplatí. Pro jednotlivé domény je možné specifikovat omezení v počtu schránek a aliasů, které jsou pak pro správce domén závazné.

Jednotlivé účty jsou samozřejmě vázány ke konkrétním doménám. Autentifikační údaje tedy tvoří login, což je celý e-mail včetně doménové části, a heslo, které je definováno v příslušné databázové tabulce spravované Postfixadminem.

Operace realizované v prostředí Postfixadminu se logují a jak hlavní správce, tak správci domén mají přehled, které operace byly kdy provedeny.

Postfixadmin umožňuje také nadefinovat automatické stahování pošty z cizích POP3/IMAP serverů do konkrétních poštovních schránek. Tato funkčnost je však vázána na příslušná nastavení, která v tomto seriálu probrána nebudou. V odkazech pod článkem však naleznete návod, jak tuto funkčnost zprovoznit.

Tímto bych tento díl ukončil. Propojení s Postfixem a Dovecotem bude probráno v příštím dílu.

## Správa linuxového serveru: Propojení Postfixadminu s Postfixem a Dovecotem a autentifikace virtuálních účtů

Minulý díl načal tvorbu řešení správy více domén na poštovním serveru za asistence webové aplikace Postfixadmin. V tomto dílu se dozvíte, jak Postfixadmin propojit s Postfixem a Dovecotem a jak vyřešit SMTP autentifikaci virtuálních účtů.

### Postfixadmin

Problematika instalace a konfigurace nástroje Postfixadmin byly probrány v minulém dílu. Pokud jste jej tedy nečetli, začněte určitě jím.

### Propojení s Postfixem

Propojení s Postfixem je popsáno v /usr/share/doc/postfixadmin/DOCUMENTS/POSTFIX_CONF.txt.gz. Doporučuji jej přinejmenším konzultovat, zejména pokud používáte novější verzi Postfixadminu. Tento návod by měl být aktuální pro verzi 2.3.5 a pro Debian Squeeze.

Do /etc/postfix/main.cf vložte následující řádky:

```
virtual_mailbox_domains = proxy:mysql:/etc/postfix/sql/mysql_virtual_domains_maps.cf
virtual_alias_maps =
proxy:mysql:/etc/postfix/sql/mysql_virtual_alias_maps.cf,
proxy:mysql:/etc/postfix/sql/mysql_virtual_alias_domain_maps.cf,
proxy:mysql:/etc/postfix/sql/mysql_virtual_alias_domain_catchall_maps.cf
virtual_mailbox_maps =
proxy:mysql:/etc/postfix/sql/mysql_virtual_mailbox_maps.cf,
proxy:mysql:/etc/postfix/sql/mysql_virtual_alias_domain_mailbox_maps.cf
virtual_mailbox_base = /var/mail/vmail
virtual_minimum_uid = 8
virtual_transport = virtual
virtual_uid_maps = static:8
virtual_gid_maps = static:8
local_transport = local:$myhostname virtual
local_recipient_maps = proxy:unix:passwd.byname $alias_maps $virtual_mailbox_maps
```

**Pozor!** Domény spravované Postfixadminem nesmí být uvedené v mydestination! Konfigurace naznačená výše by vám měla umožnit používat jak lokální doručování pro domény uvedené v mydestination, tak virtuální doručování pro virtuální účty domén spravovaných Postfixadminem.

Pokud se na obsah výše uvedených proměnných podíváte podrobněji, zjistíte, že se tato konfigurace odkazuje na řadu dalších souborů, které vytvoříte níže, a to v adresáři /etc/postfix/sql. Na názvech i cestách k jednotlivým souborům nezáleží, avšak musí se shodovat s názvy a cestami uvedenými v konfiguraci Postfixu.

Princip tohoto nastavení spočívá v definici databázových přístupových údajů a jednotlivých databázových dotazů, kterými Postfix získá potřebné informace. To samozřejmě znamená, že od této chvíle bude fungování Postfixu závislé na dostupnosti databázového serveru.

Vytvořte soubor /etc/postfix/sql/mysql_virtual_alias_maps.cf s následujícím obsahem:

```
user = uzivatel
password = heslo
hosts = localhost
dbname = nazev_databaze
query = SELECT goto FROM alias WHERE address='%s' AND active = '1'
```

Proměnné user, password a dbname si upravte podle toho, jak jste nastavili Postfixadmin při instalaci. Pokud nevíte, podívejte se do souboru /etc/postfixadmin/dbconfig.inc.php, kde jsou správné hodnoty uvedeny. Tyto hodnoty je třeba upravit v každém souboru!

Vytvořte soubor /etc/postfix/sql/mysql_virtual_alias_domain_maps.cf s následujícím obsahem:

```
user = uzivatel
password = heslo
hosts = localhost
dbname = nazev_databaze
query = SELECT goto FROM alias,alias_domain WHERE alias_domain.alias_domain = '%d' and alias.address = CONCAT('%u', '@',
alias_domain.target_domain) AND alias.active = 1 AND alias_domain.active='1'
```

Opět si přizpůsobte výše uvedené proměnné. Poté vytvořte soubor/etc/postfix/sql/mysql_virtual_alias_domain_catchall_maps.cf s následujícím obsahem:

```
user = uzivatel
password = heslo
hosts = localhost
dbname = nazev_databaze
query  = SELECT goto FROM alias,alias_domain WHERE alias_domain.alias_domain = '%d' and alias.address = CONCAT('@',
alias_domain.target_domain) AND alias.active = 1 AND alias_domain.active='1'
```

Nezapomeňte si upravit proměnné. Následně vytvořte soubor/etc/postfix/sql/mysql_virtual_domains_maps.cf s následujícím obsahem:

```
user = uzivatel
password = heslo
hosts = localhost
dbname = nazev_databaze
query          = SELECT domain FROM domain WHERE domain='%s' AND active = '1'
```

Po úpravě proměnných vytvořte soubor /etc/postfix/sql/mysql_virtual_mailbox_maps.cf s následujícím obsahem:

```
user = uzivatel
password = heslo
hosts = localhost
dbname = nazev_databaze
query = SELECT maildir FROM mailbox WHERE username='%s' AND active = '1'
```

Po úpravě proměnných vytvořte soubor /etc/postfix/sql/mysql_virtual_alias_domain_mailbox_maps.cf s následujícím obsahem:

```
user = uzivatel
password = heslo
hosts = localhost
dbname = nazev_databaze
query = SELECT maildir FROM mailbox,alias_domain WHERE alias_domain.alias_domain = '%d' and mailbox.username = CONCAT('%u', '@',
alias_domain.target_domain) AND mailbox.active = 1 AND alias_domain.active='1'
```

Následně restartujte Postfix:

```
service postfix restart
```

Abyste propojení otestovali, vytvořte v Postfixadminu doménu a poštovní schránku. Pokud neodškrtnete příslušné políčko, měl by být na danou adresu automaticky odeslán uvítací e-mail, který de facto danou poštovní schránku vytvoří (a to ve /var/mail/vmail). Můžete zkontrolovat, zda se příslušné adresáře vytvořily a zpráva přistála v příslušném maildiru v tomto umístění (maildir je tvořen třemi adresáři, cur, new atmp, jednotlivé e-maily jsou uloženy v samostatných souborech, nové e-maily směřují do adresáře new). Náhledem do logu (/var/log/mail.log) můžete odhalit problém, pokud se to Postfixu nepodaří. Doplním, že při ladění urychlíte opětovné pokusy o doručení příkazem pro vyprázdnění poštovní fronty:

```
postqueue -f
```

### Propojení s Dovecotem

Nejprve upravte soubor /etc/dovecot/dovecot.conf následujícím způsobem:

```
mail_location = maildir:/var/mail/vmail/%u/

auth default {
mechanisms = plain
```

```
userdb sql {
# Path for SQL configuration file, see doc/dovecot-sql-example.conf
args = /etc/dovecot/dovecot-mysql.conf
}
passdb sql {
# Path for SQL configuration file, see doc/dovecot-sql-example.conf
args = /etc/dovecot/dovecot-mysql.conf
}
}

first_valid_uid = 8
```

Při úpravách berte na vědomí, že konfigurační soubor Dovecotu, dovecot.conf, je hojně komentovaný a obsahuje prakticky všechny výše uvedené konstrukce a proměnné, ovšem s jinými hodnotami. Proto si dejte pozor, abyste někde něco neměli definované dvakrát. Následně vytvořte soubor /etc/dovecot/dovecot-mysql.conf s následujícím obsahem:

```
connect = host=localhost dbname=nazev_databaze user=uzivatel password=heslo
driver = mysql
default_pass_scheme = MD5-CRYPT

password_query = SELECT username AS user,password FROM mailbox WHERE username = '%u' AND active='1'
user_query = SELECT maildir, 8 AS uid, 8 AS gid FROM mailbox WHERE username = '%u' AND active='1'
```

I zde je třeba provést úpravu proměnných dbname, user a password hodnotami specifikovanými v/etc/postfixadmin/dbconfig.inc.php. Povšimněte si části dotazu user_query, který obsahuje hodnoty UID a GID, pod kterými se má přistupovat k poště. Číslo 8 odpovídá uživateli a skupině mail v distribuci Debian. Aby Dovecot s takovým UID neodmítl pracovat, je třeba uvést stejné nebo nižší číslo v proměnnéfirst_valid_uid (viz výše).

**SMTP autentifikace a virtuální účty**

Jelikož pro SMTP autentifikaci nejspíše využíváte SASL a povolujete vzdálený přístup k SMTP serveru prostřednictvím permit_sasl_authenticated v nastavení Postfixu (viz dřívější díly), výše uvedené úpravy vám neumožní odesílat poštu z virtuálních účtů (autentifikace vám selže). I pro tuto záležitost existuje řada možných řešení. Já uvedu řešení spočívající v autentifikaci prostřednictvím lokálního IMAP serveru.

Upravte/etc/default/saslauthd následujícím způsobem:

```
MECHANISMS="rimap"
MECH_OPTIONS="localhost"
OPTIONS="-c -r -O localhost -m /var/spool/postfix/var/run/saslauthd"
```

A ještě upravte /etc/postfix/sasl/smtpd.conf, takto:

```
pwcheck_method: saslauthd
saslauthd_path: /var/run/saslauthd/mux
log_level: 3
mech_list: PLAIN LOGIN
auxprop_plugin: rimap
```

Poté restartujte saslauthd a postfix:

```
service saslauthd restart
service postfix restart
```

Nyní vyzkoušejte odeslat e-mail z některé z virtuálních schránek. Připomínám, že jako login je třeba použít celý e-mail včetně doménového jména, heslo pak odpovídá heslu dané poštovní schránky.

## Správa linuxového serveru: Úvod do DNS

DNS je po TCP/IP jedna z nejdůležitějších služeb na internetu. Ať chcete nebo nechcete provozovat vlastní DNS server, základy DNS potřebuje znát každý správce serverů. Tento díl seriálu osvětlí tuto problematiku.

### Rychlokurz DNS

K čemu slouží DNS? Většina z vás to asi bude vědět, ale přesto si neodpustím malé opakování. Komunikace mezi jednotlivými dvěma uzly na internetu (např. mezi klientem a serverem) je řešena protokolem IP – každý z uzlů má přidělenou IP adresu, která jej jednoznačně identifikuje (pokud není z privátního rozsahu, ale to je úplně jiný problém pro samostatný traktát) a umožní pomocí směrovačů mezi oběma uzly doručovat data z jednoho uzlu na druhý a naopak.

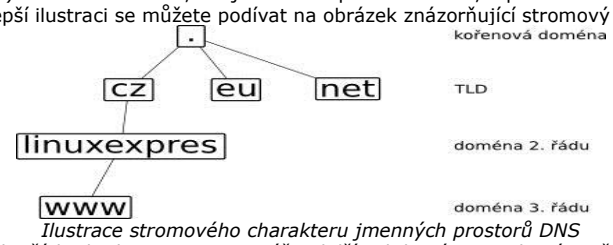
IP adresa je vhodný strojový identifikátor, ale nikoliv vhodný identifikátor pro použití člověkem. Čtyři čísla oddělená tečkou (IPv4) nebo osm skupin čtyř hexadecimálních číslic (IPv6) není snadné k zapamatování – člověk si mnohem lépe zapamatuje jméno (linuxexpres.cz) než hromadu číslic (91.214.192.101). Služba DNS zajišťuje převod jmen na IP adresy (a do jisté míry „naopak").

#### *Stromový charakter*

Jmenný systém DNS má charakter stromu. Kořenem tohoto stromu je tzv. kořenová doména, která se vyjadřuje tečkou. Pod kořenovou doménou jsou tzv. TLD, top-level domény, které se dělí na generické (com, org, net atd.), státní (cz, pl, co.uk, us atd.) a infrastrukturní (arpa). Přidělování TLD jednotlivým správcům má na starosti ICANN. Správci TLD pak, zpravidla za poplatek, umožňují registraci domén druhého (někdy i dalšího) řádu. Domény druhého řádu pak patří pod danou TLD. Zápis jednotlivých úrovní DNS jmen se provádí zprava doleva, tzn. máte-li doménu druhého řádu "linuxexpres" a TLD "cz", zápis bude vypadat takto:

```
linuxexpres.cz.
```

Tečka na konci (tedy kořenová doména) se běžně neuvádí, ale je součástí protokolu DNS, a pokud budete spravovat vlastní DNS server, v jeho konfiguraci ji uvádět budete. Pro lepší ilustraci se můžete podívat na obrázek znázorňující stromový charakter jmenného systému DNS:




*Ilustrace stromového charakteru jmenných prostorů DNS*

Pokud si zaregistrujete doménu druhého řádu, budete si moci vytvářet další subdomény, tj. domény třetího řádu, domény čtvrtého řádu atd., např.:

```
www.linuxexpres.cz
server.example.org
my.server.example.org
```

Registrace domén probíhá přes tzv. registrátory. Ti za vás vyřídí nejenom registraci dané domény, ale také zpravidla umožňují vytvořit a spravovat vaši doménu na jejich DNS serverech. Děje se tak obvykle přes webové rozhraní. Můžete si samozřejmě vytvořit vlastní DNS server a svoje domény spravovat na něm (jak, to se dozvíte v dalších dílech seriálu).

#### *Protokol DNS*

Z technického hlediska běží DNS nejčastěji na UDP, ale může používat i TCP. UDP je používáno kvůli rychlosti – jak možná víte, než mohou přes TCP proudit nějaká data, musí dojít k vytvoření spojení, k čemuž jsou potřeba tři pakety, a pak je teprve možné vznést DNS dotaz. UDP je v tomto případě rychlejší, na DNS dotaz stačí jediný paket.

Pro diagnostiku a studium DNS doporučuji nástroj *dig*, který se ukrývá v balíčku dnsutils. Na Debianu jej nainstalujete takto:

```
aptitude install dnsutils
```

Základní DNS dotaz můžete provést takto:

```
dig example.org

; <<>> DiG 9.9.1-P1 <<>> example.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10873
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.org.                   IN      A

;; ANSWER SECTION:
example.org.            172800  IN      A       192.0.43.10

;; AUTHORITY SECTION:
example.org.            172800  IN      NS      b.iana-servers.net.
example.org.            172800  IN      NS      a.iana-servers.net.

;; Query time: 879 msec
;; SERVER: 10.0.1.254#53(10.0.1.254)
;; WHEN: Tue Jul 24 17:08:10 2012
;; MSG SIZE  rcvd: 104
```

V tomto výpisu vidíte dotaz (question section), odpověď (answer section) a autoritativní servery pro daný dotaz. Všimněte si tečky za každým doménovým jménem (kořenová doména).

DNS dotazy mají charakter otázka-odpověď, nicméně je třeba tušit, koho se musíte zeptat, jakou odpověď vám poskytne a co s ní budete dále dělat. Více viz dále.

#### *Hierarchie DNS serverů*

Zadali jste jednoduchý DNS dotaz a přišla vám odpověď. Jakým způsobem ale byla získána? DNS je hierarchický systém, tzn. neexistuje jeden server, kterého byste se na cokoliv zeptali, a on by znal odpověď. DNS serverů je řada a každý z nich obsahuje jiné informace. Na počátku hierarchie DNS stojí tzv. root servery. Ty obsahují informaci o kořenové doméně a tuší, kde jsou autoritativní servery jednotlivých TLD. Nic víc netuší. Autoritativní servery jednotlivých TLD obsahují informace o autoritativních serverech jednotlivých domén druhého řádu.

Uvažte doménové jméno wikipedia.org. Abyste toto jméno přeložili na IP adresu, musíte se nejprve zeptat některého z kořenových serverů. Jak zjistíte jeho IP adresu? Nijak – seznam kořenových serverů v podobě jednotlivých IP adres musíte mít někde uložený.

Kořenový server sice netuší, jakou IP adresu má wikipedia.org, ale poskytne vám seznam autoritativních serverů pro danou TLD, tedy org. Tyto servery spravuje majitel dané TLD. Vy si některý z nich vyberete a dotážete se ho na wikipedia.org. Konkrétní IP adresu sice zpravidla vědět nebude, ale dá vám opět seznam autoritativních serverů, tentokrát ale autoritativních pro doménu wikipedia.org. Vy si opět vyberete některého z nich, a ten vám konečně řekne, že pod jménem wikipedia.org se skrývá počítač*208.80.152.201*.

Kdybyste si to chtěli vyzkoušet, vyberte si nějaký kořenový server (seznam najdete třeba na Wikipedii) a nějakou doménu, kterou chcete přeložit (např. wikipedia.org) a proveďte:

```
dig wikipedia.org @korenovy_server
```

```
;; AUTHORITY SECTION:
org.                    172800  IN      NS      a0.org.afilias-nst.info.
;; ADDITIONAL SECTION:
a0.org.afilias-nst.info. 172800 IN      AAAA    2001:500:e::1
a0.org.afilias-nst.info. 172800 IN      A       199.19.56.1
```

Výpis je zkrácený, nicméně vidíte, že kořenový server na daný dotaz odpověděl seznamem autoritativních serverů pro top-level doménu org. Přiložil také A a AAAA záznamy daných DNS serverů. Jeden z nich si vyberte a dotaz opakujte:

```
dig wikipedia.org @199.19.56.1

;; AUTHORITY SECTION:
wikipedia.org.          86400   IN      NS      ns1.wikimedia.org.

;; ADDITIONAL SECTION:
ns0.wikimedia.org.      86400   IN      A       208.80.152.130
```

Ani zde se nedočkáte přímé odpovědi, nicméně nyní už víte, které DNS servery spravují doménuwikipedia.org. Některý z nich si vyberte a dotaz opakujte:

```
dig wikipedia.org @208.80.152.130

;; ANSWER SECTION:
wikipedia.org.          3600    IN      A       208.80.152.201
```

A je hotovo, vrátil se vám A záznam pro wikipedia.org ukazující na IP adresu *208.80.152.201*.

***Typy DNS záznamů a zóny***

Zóna je část doménového jména, jehož správa byla někomu svěřena. Vy se asi nejčastěji setkáte s doménou druhého řádu, např. example.org. Zóna obsahuje jednotlivé DNS záznamy (resource records). Tyto záznamy mohou být různého typu. Nejčastější je A záznam, který mapuje jméno (např. example.org) na konkrétní IPv4 adresu. Pro mapování jména na IPv6 adresu slouží AAAA záznam. Ptáte-li se tedy na IP adresu jména example.org, ptáte se vlastně na A záznam. Doména může také obsahovat záznam typu MX (mail exchange), který označuje jednotlivé poštovní servery pro danou doménu a jejich priority. Častý je také záznam typu CNAME, což je v podstatě alias mapující jméno na jméno tzn. např. core.example.orgmůže být aliasem pro www.example.org. Důležitý je také záznam typu NS, kterým můžete delegovat subdoménu na jiný DNS server. Kořenové servery obsahují výhradně záznamy typu NS pro jednotlivé TLD.

***DNS resolver a kešování***

Pokud chcete vyřešit DNS dotaz, sami víte, že se přímo nemusíte ptát kořenových DNS serverů a řešit, koho se máte ptát dál. Prostě zadáte jméno třeba do prohlížeče a váš počítač DNS dotaz vyřeší a spojí se s daným serverem. Jak jej ale vyřeší? Operační systémy mají jednu vestavěnou funkčnost pro řešení DNS dotazů, říká se jí resolver. Resolver se v GNU/Linuxu nastavuje v konfiguračním souboru /etc/resolv.conf. Tento soubor obsahuje seznam DNS serverů, kterých se bude resolver ptát. Tyto servery vám obvykle přidělí ISP, ale můžete využít i některé veřejně dostupné DNS servery (nebo si zřídit vlastní, ale o tom až někdy příště). Tento soubor obvykle vypadá nějak takto:

```
nameserver 1.2.3.4
nameserver 4.3.2.1
```

Dobrá. Takže zadáte do prohlížeče nějaké jméno. Prohlížeč se zeptá resolveru. Resolver se nejprve podívá do své vlastní keše, jestli tam to jméno už nemá. Pokud ano, vrátí ho prohlížeči a proces je ukončen. Pokud ne, zeptá se některého z DNS serverů specifikovaných v /etc/resolv.conf. Ten se také nejprve podívá do vlastní keše a vyhledává pouze informace, které v keši nemá. Resolveru nakonec přijde odpověď, kterou předá procesu, jenž se ptal, a záznam si uloží do své keše pro případnou budoucí potřebu.

Všechny DNS záznamy mají časově omezenou platnost, která řídí, jak dlouho si mají kešovací DNS servery a resolver informace ponechat. Po uplynutí této doby se záznam z keše vymaže.

***Typy DNS serverů***

DNS servery se dají členit dle různých kritérií. Výše jsem několikrát použil pojem „autoritativní DNS server". Tímto pojmem je označován server, který obsahuje původní informace o dané zóně (informace z první ruky). Pokud se zeptáte kešovacího DNS serveru, jsou to informace z druhé ruky. Autoritativní servery spravuje majitel dané zóny.

V redukovaných výpisech výše to asi nebylo patrné, ale pokud si nějaké dotazy budete zkoušet, zjistíte, že autoritativních DNS serverů je vždy více. Je to redundance pro případ, že by některý selhal. Jeden z nich je vždy primární a ostatní sekundární. Sekundární přebírají informace z primárních.

Tím bych tento díl ukončil. V dalších se budeme problematikou DNS zabývat hlouběji.

## Správa linuxového serveru: DNS a DHCP server dnsmasq

Tento díl pokračuje v problematice DNS, věnuje se rekurzivním dotazům, PTR záznamům a DNSSEC. Představuje také server dnsmasq, který poslouží jako na zdroje nenáročný a snadno konfigurovatelný kešovací DNS a DHCP server.

### Rekurzivní dotazy

Většina z vás bude jistě tušit (zejména po přečtení předchozího dílu), že pomocí DNS můžete získat ke jménu (např. wikipedia.org) konkrétní IP adresu (např. 208.80.152.201). DNS ale také podporuje „obrácené", rekurzivní dotazy, tedy takové dotazy, které vám vrátí jméno ke konkrétní IP adrese. Využívají se k tomu záznamy typu PTR a speciální doména in-addr.arpa pro IPv4, ip6.arpa pro IPv6.

Zkuste si to na nějakém příkladu. Můžete využít nástroj dig, který byl představen v minulém díle. Nejprve získejte IP adresu k nějakému jménu (tedy A záznam), třeba wikipedia.org:

```
dig wikipedia.org

;; QUESTION SECTION:
;wikipedia.org.                 IN      A

;; ANSWER SECTION:
wikipedia.org.          3516    IN      A       208.80.152.201
```

Chcete-li provést rekurzivní DNS dotaz pomocí nástroje dig, použijte volbu -x, takto:

```
dig -x 208.80.152.201

;; QUESTION SECTION:
;201.152.80.208.in-addr.arpa.   IN      PTR

;; ANSWER SECTION:
201.152.80.208.in-addr.arpa. 3361 IN    PTR     wikipedia-lb.pmtpa.wikimedia.org.
```

Z odpovědi na tento dotaz by mělo být jasné, jak je vlastně rekurzivní dotaz realizován. Hledá se záznam typu PTR ke jménu 201.152.80.208.in-addr.arpa, přičemž toto jméno obsahuje číselnou reprezentaci IP adresy, ovšem jednotlivé oktety jsou uvedeny v opačném pořadí. Jak víte, systém DNS je tvořen stromovou strukturou, která má počátek na konci (top-level doména), a pak se dále větví do domén druhého, třetího a dalších řádů. Každá větev nebo „podvětev" tohoto stromu pak může být delegována jinému subjektu.

Jak je patrné z příkladu výše, PTR záznamy nemusí mít stejnou hodnotu jako A záznamy. Dlužno dodat, že PTR záznamů pro jednu IP adresu může být více, nicméně není to obvyklé, často z obav, že nebude správně fungovat software očekávající pouze jedno jméno na rekurzivní dotaz.

PTR záznamy jsou důležité zejména v oblasti pošty, jelikož některé poštovní servery předpokládají a očekávají shodu A záznamu s PTR záznamem. Pokud se záznamy shodovat nebudou, pošta může být odmítnuta, klasifikována jako spam nebo se zvýší skóre nástrojů typu spamassassin.

PTR záznamy obvykle nastavujete u svého ISP, který vám buď deleguje danou DNS větev na váš vlastní DNS server, nebo vám nastaví PTR záznamy na svém.

### Dnsmasq: Jednoduchý DNS server

Dnsmasq je vynikající nástroj, pokud potřebujete rychle nasadit kešující DNS (anebo také DHCP) server pro malou síť (dle webu projektu zvládá i tisíc klientů). Dnsmasq není resolver, tzn. neprovádí vyhledávání neznámých domén sám, nýbrž jen předává dotazy místním DNS resolverům (viz /etc/resolv.conf). Vyřešené dotazy kešuje, což snižuje zátěž hlavních DNS resolverů a také mírně zvyšuje výkon. Můžete samozřejmě přidávat vlastní DNS záznamy pro místní nebo speciální domény.

Dnsmasq umí fungovat také jako jednoduchý DHCP server, IP adresy můžete přidělovat staticky nebo dynamicky. Jeho předností je nenáročnost na paměť a na konfiguraci.

Instalaci nástroje dnsmasq provedete v Debianu klasickým způsobem, instalací příslušného balíčku z oficiálních repozitářů:

```
aptitude install dnsmasq
```

#### *Konfigurace*

Hlavním konfiguračním souborem je /etc/dnsmasq.conf. Tento konfigurační soubor je bohatě komentovaný, takže není problém dnsmasq rychle nastavit podle vašich představ. První věc, kterou byste měli upravit, je specifikace rozhraní, kde má dnsmasq naslouchat. Z bezpečnostních důvodů se nehodí, aby byl dostupný z internetu (ve výchozím stavu naslouchá na všech rozhraních):

```
interface=eth1
```

Rozhraní můžete specifikovat více:

```
interface=eth2
interface=eth3
```

#### *DNS*

Kešování DNS a forwarding fungují automaticky, k tomu není třeba nic nastavovat. Můžete se sami přesvědčit tím, že se zeptáte na nějakou doménu :

```
dig www.wikipedia.org @127.0.0.1
```

Dnsmasq využívá vaše resolvery (viz /etc/resolv.conf) a odpovědi kešuje. Integruje také informace z/etc/hosts, což velmi usnadňuje pojmenování místních počítačů – odpadá nutnost tvořit zónový soubor. Například, pokud do /etc/hosts vložíte záznam:

```
10.0.1.15 kremilek.local
```

...můžete se po restartu programu dnsmasq rovnou přesvědčit, že od něj dostanete správnou odpověď (tedy, správnou dle /etc/hosts):

```
dig @127.0.0.1 kremilek.local

;; ANSWER SECTION:
kremilek.local.         10800   IN      A       10.0.1.15
```

Výchozí hodnota pro velikost keše je sto padesát záznamů, což není mnoho. Chcete-li kešovat více záznamů, specifikujte příslušnou hodnotu v direktivě cache-size:

```
cache-size=500
```

Vlastní DNS záznamy můžete vkládat nejenom do /etc/hosts, ale také do konfigurace samotné:

```
host-record=laptop,laptop.local,10.0.1.60,1234::100
```

Výše uvedená řádka vytvoří A záznam pro jména laptop a laptop.local s IP adresou 10.0.1.60 a AAAA (IPv6) záznam s IPv6 adresou 1234::100.

Potřebujete-li vložit vlastní MX záznam, můžete použít mx-host:

```
mx-host=example.org,postovniserver.local,10
```

Výše uvedený řádek nastaví MX záznam pro doménu example.org o váze 10 s poštovním serverempostovniserver.local.

Podobným způsobem můžete vytvářet i PTR nebo SRV záznamy. Více viz manuálová stránka a komentáře v konfiguračním souboru. Po úpravě konfigurace je třeba dnsmasq restartovat:

```
service dnsmasq restart
```

#### *DHCP*

Funkci DHCP serveru má dnsmasq ve výchozím stavu vypnutou. Pro její zprovoznění postačí specifikovat rozsah adres pro DHCP:

```
dhcp-range=10.0.1.50,10.0.1.150,12h
```

Tato řádka nastaví dynamické přidělování adres v rozsahu 10.0.1.50 až 10.0.1.150, přičemž přidělená IP adresa bude „pronajatá" dvanáct hodin.

Podobně jako v případě direktivy interface můžete dhcp-rangespecifikovat vícekrát, pokaždé pro jiný rozsah:

```
dhcp-range=10.0.1.50,10.0.1.150,12h
dhcp-range=10.0.5.10,10.0.5.50,12h
```

IP adresy můžete specifikovat i staticky, pro konkrétní MAC síťové karty:

```
dhcp-host=11:22:33:44:55:66,10.10.10.10
```

Tato řádka přidělí MAC adrese 11:22:33:44:55:66 IP adresu 10.10.10.10.

Konkrétní rozsahy můžete přidělovat i konkrétním síťovým rozhraním:

```
dhcp-range=eth1,10.0.1.50,10.0.1.150,12h
```

```
dhcp-range=eth2,10.0.5.10,10.0.5.50,12h
```

Potřebujete-li specifikovat některé další volby jako router apod., můžete použít dhcp-option:

```
dhcp-option=option:router,10.10.10.10
dhcp-option=option:ntp-server,10.10.10.20,10.10.10.30
```

Výše uvedená řádka sdělí DHCP klientům, že router je na adrese 10.10.10.10 a NTP servery jsou na10.10.10.20 a 10.10.10.30. Další příklady naleznete v konfiguračním souboru a v manuálu.

## DNSSEC

DNS je velmi starý protokol, který nebyl navržen s ohledem na bezpečnost (což samozřejmě neplatí jen o DNS). Existuje možnost podvržení DNS záznamů, které není možné spolehlivě zabránit, což otevírá prostor pro známý man-in-the-middle útok. Podaří-li se útočníkovi podvrhnout odpověď na váš DNS dotaz, dojde k „otrávení" vaší DNS keše podvrženým záznamem (proto se tyto útoky označují jako DNS cache poisoning). Snad nemusím zdůrazňovat, jaké důsledky může mít podvržení DNS záznamů – místo komunikace se svou bankou můžete komunikovat s útočníkem, který pak může zaútočit přímo na vaše bankovní konto.

Obranou vůči těmto útokům je DNSSEC, což je systém, který DNS záznamy digitálně podepisuje (používá se asymetrické šifrování), a DNS resolvery podpisy kontrolují. Digitální podpisy mají svou hierarchii (tzv.chain of trust model), která začíná u kořenové zóny (počáteční bod hierarchie se označuje jako „trust anchor", kotva důvěry – její podpis byste měli pečlivě ověřit, jelikož s jeho důvěryhodností stojí a padá důvěryhodnost celého ověřování). Tento systém je poměrně komplexní a komplikovaný – více o principech fungování DNSSEC se dozvíte v odkazech pod článkem.

DNSSEC je třeba implementovat na straně resolvujících DNS serverů, které využíváte. Nepodporuje-li tedy váš ISP DNSSEC, můžete si postavit vlastní resolver se zprovozněným DNSSECem. Zda váš ISP používá DNSSEC (resp. zda vámi použité resolvující DNS servery používají DNSSEC), zjistíte dotazem na špatně podepsanou doménu, kterou je např. rhybar.cz:

```
dig www.rhybar.cz

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56068

;; QUESTION SECTION:
;www.rhybar.cz.                 IN      A

;; ANSWER SECTION:
www.rhybar.cz.          504     IN      A       217.31.205.51
```

Pokud vám na tento dotaz vrátí server bezchybnou odpověď (status: NOERROR), pak DNSSEC implementovaný nemá a vy jste zranitelní. Dostane-li se vám odpovědi se statusem SERVFAIL:

```
dig www.rhybar.cz

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 18561
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.rhybar.cz.                 IN      A
```

...pak je všechno v pořádku a DNSSEC je aktivní. Jednoduchý test přímo z webového prohlížeče můžete realizovat třeba na stránce www.dnssec.cz. Jak si postavit vlastní resolvující a DNSSEC validující server, se dozvíte v příštím díle.

## Správa linuxového serveru: Unbound a Bind

Tento díl se věnuje stavbě vlastního DNS serveru, konkrétně pak rekurzivního a DNSSEC validujícího serveru Unbound, a také víceúčelového serveru Bind.

Jak už bylo řečeno dříve, různé DNS servery realizují různou funkčnost. V zásadě existují dva základní typy, rekurzivní a autoritativní. **Rekurzivní DNS servery** realizují pro své klienty vyhledávání v DNS stromu a poskytují jim odpovědi na jejich dotazy. Servery tohoto typu vám bývají poskytovány vaším ISP. Rekurzivní DNS servery provádějí také kešování odpovědí na dotazy.

Kromě rekurzivních serverů existují také tzv. **autoritativní DNS servery**, což jsou servery, které publikují DNS záznamy určité části DNS stromu, jejíž správa jim byla přidělena. Kupříkladu, pokud byste si zaregistrovali doménu *example.org*, měli byste kontrolu nad tím, které DNS servery budou obsahovat její autoritativní informace. Na tyto DNS servery by se pak obracely rekurzivní DNS servery, které od nich získají odpověď na DNS dotazy týkající se domény *example.org* a jejích subdomén.

Různé DNS servery realizují různou funkčnost. Server *dnsmasq*, probraný v minulém díle, je ve své podstatě pouze kešující DNS server, který nevyřizuje dotazy, pouze je přeposílá (forwarduje) definovaným rekurzivním DNS serverům. Unbound je primárně rekurzivní a kešující DNS server, i když do jisté míry se dá použít i jako autoritativní server. Bind je univerzální DNS server schopný fungovat jako rekurzor i jako autoritativní server. Jedná se o nejznámější DNS server vůbec. Jsou i čistě autoritativní servery, které neumí pracovat jako rekurzory a zaměřují se pouze na publikování autoritativních DNS informací. Příkladem může být KnotDNS, server z dílen CZ.NIC.

Připomínám, že výchozí port pro službu DNS je 53 a komunikace může probíhat jak přes UDP, tak přes TCP.

### Unbound

Unbound je především rekurzivní a kešující DNS server. Je schopen provádět DNSSEC validaci, ale neprovádí ji ve výchozím stavu (alespoň v Debianu). Problematikou DNSSEC se zabýval minulý díl seriálu.

Instalaci serveru Unbound realizujete velmi jednoduše:

```
aptitude install unbound
```

Po instalaci se server rovnou rozběhne a začne fungovat. Jeho funkčnost si můžete snadno ověřit:

```
dig @localhost example.org
```

#### *DNSSEC*

Jako první by bylo dobré zapnout DNSSEC validaci. K tomu potřebujete do hlavního konfiguračního souboru `/etc/unbound/unbound.conf` přidat informace o souboru obsahujícím klíč kořenové zóny (fungující jako „trust anchor“, tedy „kotva důvěry“):

```
auto-trust-anchor-file: "/etc/unbound/root.key"
```

Tento soubor však ve výchozí instalaci neexistuje a je třeba jej vytvořit. Za tímto účelem je nutné si příslušný klíč opatřit a ověřit. Opatřit si jej můžete stažením následujícího souboru:

```
https://data.iana.org/root-anchors/root-anchors.xml
```

Jedná se o XML soubor s potřebnými hodnotami. Tyto hodnoty vložte do souboru `/etc/unbound/root.key`do jediné řádky, a to dle následujícího klíče:

```
. IN DS <KeyTag> <Algorithm> <DigestType> <Digest>
```

Výsledek může vypadat nějak takto:

```
.IN DS 19036 8 2 49AAC11D7B6F6446702E54A1607371607A1A41855200FD2CE1CDDE32F24E8FB5
```

Po uložení souboru Unbound restartujte:

```
service unbound restart
```

Následně zkontrolujte, že DNSSEC validace funguje, např. takto:

```
dig +dnssec org @localhost
```

Je-li DNSSEC aktivní, měli byste vidět příznak „AD“ v odpovědi:

```
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1
```

#### *Přizpůsobení nastavení Unboundu*

Jak je z postupu výše patrné, soubor s konfigurací Unboundu je `/etc/unbound/unbound.conf`. Tento soubor je bohatě komentovaný, doporučuji si jej tedy projít a upravit podle vašich potřeb. Základní dvě věci pro úpravu jsou v první řadě rozhraní a IP adresy, na kterých Unbound poslouchá (ve výchozím stavu naslouchá pouze na lokální smyčce, tedy nikoliv na síti), a ve druhé řadě je to řízení přístupu, tj. kdo má právo na vyřizování DNS dotazů.

Specifikace rozhraní, popřípadě IP adres, na kterých Unbound naslouchá, se realizuje takto:

```
interface: 1.2.3.4
interface: 1.2.3.4@5000
interface: 0.0.0.0
interface: ::0
```

První řádka zajistí, že Unbound bude naslouchat na IP adrese 1.2.3.4 na standardním portu. Druhá řádka ukazuje, jak byste Unboundu sdělili, aby naslouchal na nestandardním portu (v tomto případě portu 5000). Třetí řádka zajistí, že Unbound bude naslouchat na všech IPv4 rozhraních a adresách. Čtvrtá pak provede totéž, ale v rámci protokolu IPv6.

Druhou podstatnou oblastí je řízení přístupu. Ve výchozím stavu je totiž veškerý přístup zakázán, kromě místní smyčky (localhostu).

```
access-control: 10.0.1.15 refuse
access-control: 10.0.1.0/24 allow
```

Syntaxe je zde na první pohled jasně patrná – v první řádce se zamítá přístup počítači 10.0.1.15, druhá řádka pak povoluje přístup ze sítě 10.0.1.0/24.

Unbound umožňuje i tvorbu místních zón, popřípadě úpravu odpovědí na specifické dotazy. Místní zónu můžete snadno a rychle vytvořit takto:

```
local-zone: "local." static
local-data: "typhoon.local. IN A 10.0.1.27"
```

První řádek definuje doménu „local“, druhý pak vytváří A záznam pro počítač 10.0.1.27 se jménem*typhoon.local*.

#### *Nastavení Unboundu jako výchozího rekurzoru pro server*

Máte-li funkční rekurzivní DNS server, můžete pro příslušný počítač zajistit, aby DNS dotazy vyřizoval přímo místní rekurzor (na místo DNS serverů poskytovatele). To zajistíte úpravou `/etc/resolv.conf` a vložením následujícího záznamu na první řádek:

```
nameserver 127.0.0.1
```

Tato úprava zajistí, že se běžící aplikace budou ptát nejprve místního DNS serveru, který bude odpovídat rychleji než servery vašeho ISP. Nevyřadíte-li ostatní záznamy v resolv.conf, zajistíte tak „fallback“ na některý z dalších serverů, pokud bude váš místní DNS server mimo provoz.

### Bind

Bind je univerzální DNS server, umí pracovat jako rekurzor a vyřizovat DNS dotazy (podpora DNSSEC je samozřejmostí), ale umí pracovat také jako autoritativní server.

Jeho instalaci provedete opět velmi jednoduše:

```
aptitude install bind9
```

Po instalaci se Bind rozběhne a začne fungovat jako rekurzivní DNS server, což si opět můžete snadno ověřit:

```
dig @localhost example.org
```

#### *DNSSEC*

Základem je opět opatřit si klíč ke kořenové zóně. Tento klíč můžete získat DNS dotazem:

```
dig . dnskey

.                       172758  IN      DNSKEY  256 3 8 AwEA
AbW4qUZUxSRqUntM9u0pvmkqRB9Z+WRPghllsekdgp8ksT5bwRBE3xwVWJJp
JgVYGvFGgLIutrGyZDJVLQX+tu+qe6HJbA8XRZsL2aA6e4MZeD4TOUIIH/c
Vlof3y4gFibjwzuuondVku9ia2MSRYnrBl+LMSRftBkVa4OvS+dij
.                       172758  IN      DNSKEY  257 3 8 AwEA
AagAIKlVZrpC6Ia7gEzahOR+9W29euxhJhVVLOyQbSEW0O8gcCjFFVQUTf6v
58fLjwBd0YI0EzrAcQqBGCzh/RStIoO8g0NfnfL2MTJRkxoXbfDaUeVPQuYE
```

hg37NZWAJQ9VnMVDxP/VHL496M/QZxkjf5/Efucp2gaDX6RS6CXpoY68LsvP
VjR0ZSwzz1apAzvN9dlzEheX7ICJBBtuA6G3LQpzW5hOA2hzCTMjJPJ8LbqF
6dsV6DoBQzgul0sGIcGOYl7OyQdXfZ57relSQageu+ipAdTTJ25AsRTAoub8
ONGcLmqrAmRLKBP1dfwhYB4N7knNnulq QxA+Uk1ihz0=

Ve výpisu budou patrné dva klíče, vy chcete ten delší (257). Klíč a hodnoty převeďte do následující podoby:

managed-keys {
"." initial-key 257 3 8
"AwEAAagAIKlVZrpC6Ia7gEzahOR+9W29euxhJhVVLOyQbSEW0O8gcCjF
FVQUTf6v58fLjwBd0YI0EzrAcQqBGCzh/RStIoO8g0NfnfL2MTJRkxoX
bfDaUeVPQuYEhg37NZWAJQ9VnMVDxP/VHL496M/QZxkjf5/Efucp2gaD
X6RS6CXpoY68LsvPVjR0ZSwzz1apAzvN9dlzEheX7ICJBBtuA6G3LQpz
W5hOA2hzCTMjJPJ8LbqF6dsV6DoBQzgul0sGIcGOYl7OyQdXfZ57relS
Qageu+ipAdTTJ25AsRTAoub8ONGcLmqrAmRLKBP1dfwhYB4N7knNnulq
QxA+Uk1ihz0=";
};

A tu vložte třeba do souboru /etc/bind/named.conf.managed. Následně vložte potřebnou *include* direktivu do/etc/bind/named.conf:

include "/etc/bind/named.conf.managed";

Toto samo o sobě by mělo stačit (DNSSEC validace je povolená, ale fungovat začne teprve po výše uvedené operaci). Bind tedy restartujte:

service bind9 restart

A následně použijte stejný postup uvedený výše pro ověření, že DNSSEC validace funguje.

Pozor! Opět zde platí, že výše uvedený klíč si musíte v každém případě nějakým způsobem ověřit, protože na něm stojí celá bezpečnost ověřování DNSSEC.

***Přizpůsobení nastavení Bindu***

Bind se umí chovat úplně stejně jako *dnsmasq*, tedy jako forwardující (přeposílá dotazy na jiný server) a kešující DNS server. Chcete-li ho takto nastavit, vložte do /etc/bind/named.conf.options, mezi složené závorky direktivy *options*, následující řádky (servery, které se mají dotazovat, se specifikují v proměnné*forwarders*):

forward only;
forwarders { 1.2.3.4; 4.3.2.1;};

Nastavení síťových rozhraní, kde má Bind naslouchat, se realizuje následovně (stejný soubor, stejné umístění jako v příkladu výše):

listen-on-v6 { any; };
listen-on { 127.0.0.1; 10.0.1.254; };

První řádka specifikuje naslouchání na IPv6 adresách. Klíčové slovo *any* zajistí, že se bude naslouchat na všech. Druhá řádka řídí naslouchání na IPv4, kde jsou specifikovány v tomto případě dvě adresy, adresa místní smyčky (localhostu) a privátní adresa *10.0.1.254*.

allow-query { 10.0.1.0/24; };

Proměnná *allow-query* definuje, kdo se smí serveru dotazovat. V tomto příkladu se serveru smí dotazovat síť *10.0.1.0/24*.

allow-recursion { 10.0.1.0/24; };

Tato proměnná definuje, kdo smí pokládat rekurzivní dotazy. Syntax je totožná jako u *allow-querry*.

Pokud nechcete provozovat rekurzivní DNS server, rekurzi jako takovou je možné vypnout:

recursion no;

Tím bych pro tentokrát skončil. Příště bude osvětlena problematika tvorby zón a zónových souborů.

## Správa linuxového serveru: Bind v roli autoritativního serveru

Dnešní díl popisuje správu domén na vlastním DNS serveru, tedy tvorbu autoritativního DNS serveru a tvorbu zónových souborů.

Na úvod předpokládám, že máte nainstalovaný DNS server Bind (balíček bind9 v Debianu). Stejně tak předpokládám, že jste si prošli posledních pár dílů, které se věnují problematice DNS serverů.

Jak bylo řečeno minule, Bind umí zastat jak roli resolvujícího DNS serveru, který za klienty vyřizuje libovolné DNS žádosti, tak roli autoritativního serveru, který vrací autoritativní údaje o částech DNS stromu, jež jsou mu delegovány.

Pokud si zaregistrujete doménu, můžete registrátorovi sdělit, jaké jsou vaše autoritativní servery, a mít DNS informace spojené s příslušnou doménou (nebo se všemi vašimi doménami) na vlastních DNS serverech. Servery jsou potřeba alespoň dva, jeden primární a jeden sekundární. Primární poskytuje vždy aktuální autoritativní informace, sekundární přebírá ve specifikovaných intervalech data z primárního serveru a kešuje je. Pokud primární server vypadne, budou se klienti dotazovat sekundárního, dokud platnost jeho dat nevyprší.

DNS servery představují součást klíčové infrastruktury, a jako takové by měly být maximálně spolehlivé. Výpadek obou DNS serverů povede k nemožnosti klientů převádět vaše domény na IP adresy. I když budou v takové situaci vaše servery běžet, klienti se na ně nedostanou. Pokud budete provozovat vlastní DNS servery, měli byste je provozovat na spolehlivých strojích u spolehlivých poskytovatelů. Ideálně by to měly být servery ve dvou různých datacentrech (jinak hrozí, že výpadek konektivity datacentra zruší oba vaše DNS servery naráz).

### Příklad zónového souboru

Vytvořte soubor někde v úložišti konfigurace pro Bind, tedy např. /etc/bind/db.example.cz, který poslouží k uložení záznamů o dané zóně. Příklad může vypadat třeba takto:

```
$TTL    3h      ; default expiration time
@       IN      SOA     ns1.example.cz. spravce.example.cz. (
                        20120820 ; serial
                        4h       ; slave refresh
                        2h       ; slave retry interval
                        2w       ; slave data expiration
                        1h )     ; maximum caching time when lookups fail
                        ;
                @       IN      NS      ns1.example.cz.
                @       IN      NS      ns2.example.cz.


example.cz.     IN      MX      10      mujmailserver1.cz.
example.cz.     IN      MX      20      mujmailserver2.cz.
                example.cz.     IN      A       1.2.3.5
                ns1             IN      A       1.2.3.4
                ns2             IN      A       1.2.3.3
        www             IN      CNAME   example.cz.
           subdomena1           IN      A       1.2.3.4
        subdomena2          IN      CNAME   example.cz.
```

Jak je patrno, není to úplně procházka růžovým sadem, ale není to zase příliš složité. Vezměme to jedno po druhém:

- **$TTL 3h** – nastaví dobu existence (time-to-live, TTL) všech záznamů, které nemají definovanou TTL někde jinde v zónovém souboru
- **IN SOA** – definuje SOA záznam (Start of Authority), který obsahuje základní informace o doméně (viz dále) – tento záznam musí zónový soubor bezpodmínečně obsahovat
- **ns1.example.cz. spravce.example.cz.** – první údaj ve specifikaci SOA záznamu náleží DNS serveru obsahujícímu autoritativní informace o dané zóně, druhý pak e-mailu správce zóny (všimněte si tečky místo zavináče)
- **20120820** – první číslo u SOA záznamu udává sériové číslo zónového souboru – **při jakékoliv změně zónového souboru musí být povýšeno**; často se bere (jak je naznačeno v tomto případě) numerická reprezentace aktuálního data, ale můžete používat jakékoliv číslo – jen jej nezapomeňte při úpravách povyšovat
- **4h** – druhý údaj u SOA záznamu udává dobu, za kterou si sekundární (slave) server stáhne data z primárního (master) serveru – v tomto případě to jsou 4 hodiny (tzn. sekundární server si stáhne data každé čtyři hodiny)
- **2h** – třetí údaj u SOA záznamu udává dobu, za kterou se má sekundární server znovu pokusit stáhnout data z primárního, pokud mu nevyšel první pokus (bude-li tedy primární server nedostupný, sekundární bude pokusy o získání dat opakovat po dvou hodinách
- **2w** – čtvrtý údaj u SOA záznamu udává dobu, za kterou data na sekundárním serveru definitivně vyprší, pokud se mu nepodaří spojit s primárním DNS serverem; v tomto případě jsou to dva týdny
- **1h** – pátý údaj u SOA záznamu udává dobu, jak dlouho si mají kešovací DNS servery pamatovat, že nějaký záznam neexistuje
- **IN NS** – záznamy udávající autoritativní DNS servery pro danou zónu, primární a sekundární (může jich být samozřejmě i více); všimněte si, že k těmto serverům jsou doplněny jejich A záznamy; NS záznamy se uvádí na dvou místech, jednak v dané zóně, a jednak v zóně vyšší úrovně (tyto záznamy obvykle definujete u registrátora domény) – pokud DNS servery patří do stejné domény, kterou spravují (jako v tomto případě), musí být bezpodmínečně v zóně vyšší úrovně definováno jak jméno, tak IP adresa (A záznam) daného DNS serveru (tato dvojice tvoří tzv. GLUE záznam), jinak vám doména nebude fungovat (vznikne kruhová závislost)
- **IN A** – na šestém řádku zespodu je uveden hlavní A záznam pro doménu, překládající doménu na IP adresu
- **CNAME** – záznam typu CNAME je alias, tzn. *www.example.cz* bude ukazovat na A záznam *example.cz*
- **MX** – zóna obsahuje specifikaci dvou poštovních (MX, Mail eXchange) serverů, a to *mujmailserver1.cz*s prioritou 10 (vyšší) a *mujmailserver2.cz* s prioritou 20 (nižší)

Všimněte si také teček za doménovými jmény. Ty označují kořenovou zónu a musí být uváděny. Aby váš DNS server začal distribuovat záznamy z tohoto zónového souboru, musíte jej zařadit do nastavení Bindu, tzn. vložit do souboru /etc/bind/named.conf.local následující:

```
zone "example.cz" {
        type master;
        file "/etc/bind/db.example.cz";
        };
```

Je zde specifikován nejen soubor, ale také typ „master“, který určuje váš DNS server jako primární vzhledem k tomuto zónovému souboru.

Před samotným restartem Bindu je vhodné konfiguraci prověřit automatizovaným nástrojem, a vyhnout se tak situaci, kdy server znovu nenastartuje kvůli chybě v definici zóny, čímž nastane výpadek. Bind obsahuje nástroj named-checkconf, který stačí bez parametrů spustit:

```
named-checkconf
```

Dle unixové tradice, pokud tento příkaz nevrátí žádný text, znamená to, že nastavení Bindu je v pořádku, a vy jej můžete v klidu restartovat:

```
service bind9 restart
```

Nyní můžete server otestovat:

```
dig example.cz @1.2.3.4

;; QUESTION SECTION:
;example.cz.                    IN      A

;; ANSWER SECTION:
example.cz.             10800   IN      A       1.2.3.4
```

```
;; AUTHORITY SECTION:
example.cz.             10800   IN      NS      ns1.example.cz.
example.cz.             10800   IN      NS      ns2.example.cz

;; ADDITIONAL SECTION:
ns1.example.cz.         10800   IN      A       1.2.3.4
ns2.example.cz.         10800   IN      A       1.2.3.3
```

Všimněte si TTL u jednotlivých záznamů, které udává 10 800 sekund, tedy 3 hodiny, což je přesně hodnota, kterou jste specifikovali v proměnné $TTL.

Otestovat můžete i další záznamy (`dig -t mx`, `dig -t soa`) atd.

## Sekundární DNS servery

Přenos dat zóny (tzv. zónový transfer, zone transfer) je třeba na **primárním** serveru autorizovat, což můžete provést specifikací příslušné IP adresy u direktivy allow-transfer, takto:

```
zone "example.cz" {
        type master;
        file "/etc/bind/db.example.cz";
        allow-transfer { @4.3.2.1; };
};
```

Následně je třeba na sekundárním serveru vytvořit specifikaci zóny typu *slave* se specifikací primárního (*master*) serveru, takto:

```
zone "example.cz" {
        type slave;
        file "/var/cache/bind/db.example.cz";
        masters { @1.2.3.4; };
};
```

Nezapomeňte oba servery restartovat, nejprve primární, pak sekundární, a následně otestovat, jestli vše funguje.

Tím bych tento díl ukončil. Příště budou probrány reverzní záznamy a také český, pouze autoritativní server KnotDNS pocházející z laboratoří CZ.NIC.

## Správa linuxového serveru: Reverzní záznamy a Knot DNS

Tento díl popisuje problematiku reverzních záznamů a představuje autoritativní DNS server Knot z dílen CZ.NIC.

Pokud jste tak ještě neučinili, projděte si předchozí díly, které se věnovaly základům DNS i BINDu. Minulý díl probral tvorbu zónových souborů, ale nedostalo se na reverzní záznamy.

### Mechanismus delegace

Zdůrazňuji, že u všech zón platí mechanismus delegace. DNS servery v hierarchii výše se odkazují na DNS servery v hierarchii níže. Váš registrátor zajistí, že do kořenové zóny domény prvního řádu bude vložen záznam o vaší doméně druhého řádu spolu s odkazem na váš DNS server, kde budete mít definovanou příslušnou zónu a v ní řadu DNS záznamů.

Samozřejmě můžete provozovat vlastní DNS server, kde si nadefinujete vlastní doménu prvního řádu a kvanta domén druhého řádu, ale dokud na vás nezačnou odkazovat příslušné DNS servery v příslušných částech DNS stromu, nebudou mít vaše záznamy globální platnost (resolvující DNS servery je nenajdou).

U PTR záznamů platí úplně stejný princip – opět na vás musí odkazovat někdo v hierarchii výše. Podstatným rozdílem ovšem je, že PTR záznamy neobsluhují registrátoři domén, nýbrž majitelé příslušných bloků IP adres, tedy vaši ISP. Ti vám mohou (ale také nemusí) nabídnout vlastní DNS server, nebo nastavit delegaci pro určitou část PTR záznamů na váš server.

### Reverzní záznamy

Jako správci jste se téměř jistě s reverzními záznamy setkali. Zatímco klasické DNS záznamy (zejména pak A pro IPv4 a AAAA pro IPv6) překládají jména na IP adresy, reverzní záznamy dělají přesný opak. Podívejte se na běžný DNS dotaz:

```
# dig wikipedia.org

;; QUESTION SECTION:
;wikipedia.org.                 IN      A

;; ANSWER SECTION:
wikipedia.org.          3461    IN      A       208.80.152.201
```

Toto je klasický dotaz na A záznam. Zeptali jste se na to, jaká je IP adresa serveru *wikipedia.org*. Nyní vezměte onu IP adresu, nebo jakoukoliv jinou, a zeptejte se na její reverzní (PTR) záznam:

```
# dig -x 208.80.152.201

;; QUESTION SECTION:
;201.152.80.208.in-addr.arpa.   IN      PTR

;; ANSWER SECTION:
201.152.80.208.in-addr.arpa. 3337 IN    PTR     wikipedia-lb.pmtpa.wikimedia.org.
```

V odpovědi na tento dotaz přímo vidíte, na co se vlastně nástroj dig ptá. Ptá se na PTR záznam k *201.152.80.208.in-addr.arpa.* – doména *in-addr.arpa* je speciální doména určená k uchovávání PTR záznamů. Povšimněte si, že PTR záznamy mají stejný stromový charakter jako klasické DNS, přičemž směrem doprava putujete v hierarchii výše a směrem doleva níže. Proto jsou také jednotlivé oktety této IPv4 adresy zapsány v opačném pořadí. Tento mechanismus umožňuje delegaci.

Připomínám, že i když je možné uvést více PTR záznamů k jediné IP adrese, zvykem je to nedělat, zpravidla kvůli špatně napsanému softwaru, který očekává, že na dotaz na PTR záznam dostane jednu nebo žádnou odpověď.

Jak je ještě patrné v příkladu výše, PTR záznamy se nemusí shodovat s A záznamy. To je v pořádku, nicméně u poštovních serverů doporučuji vytvořit PTR záznam ve shodě s příslušným A záznamem, což o něco zvýší pravděpodobnost úspěšného doručení vaší pošty (vzhledem k opatřením proti spamu, která tuto shodu v některých případech prověřují).

### Reverzní záznamy v BINDu

Pokud byste chtěli vytvořit PTR zónu pro subnet 10.0.0.0/24, vytvořili byste zónový soubor (např. `/etc/bind/db.10.0.0`) s obsahem podobným tomuto:

```
                        $TTL    86400
0.0.10.in-addr.arpa.    IN      SOA     ns1.example.cz admin.example.cz. (
                                20120820 ; serial
                                4h       ; slave refresh
                                2h       ; slave retry interval
                                2w       ; slave data expiration
                                1h )     ; maximum caching time when lookups fail
                                ;
0.0.10.in-addr.arpa.    IN      NS      ns1.example.cz.
0.0.10.in-addr.arpa.    IN      NS      ns2.example.cz.

1.0.0.10.in-addr.arpa.  IN      PTR     webserver.example.org.
254.0.0.10.in-addr.arpa.  IN    PTR     router.example.org.
```

Jak vidíte, TTL i základní parametry zóny jsou shodné, dva NS záznamy pro autoritativní DNS servery také zůstávají, zóna pak obsahuje dva PTR záznamy pro počítače *10.0.0.1* (*webserver.example.org*) a *10.0.0.254* (*router.example.org*).

Nezapomeňte na tento zónový soubor upozornit Bind, a to editací /etc/bind/named.conf.local:

```
zone "0.0.10.in-addr.arpa" {
        type master;
        file "/etc/bind/db.10.0.0";
};
```

Před restartem Bindu zkontrolujte, že konfigurace je v pořádku:

```
named-checkconf
```

Můžete také zkontrolovat přímo danou zónu v daném zónovém souboru, a to pomocí nástroje *named-checkzone*:

```
# named-checkzone 0.0.10.in-addr.arpa. /etc/bind/db.10.0.0
zone 0.0.10.in-addr.arpa/IN: loaded serial 20120820
OK
```

Po kontrole můžete provést restart:

```
service bind9 restart
```

Nyní můžete dotaz na reverzní záznam otestovat:

```
# dig -x 10.0.0.1 @127.0.0.1

;; QUESTION SECTION:
;1.0.0.10.in-addr.arpa.         IN      PTR

;; ANSWER SECTION:
1.0.0.10.in-addr.arpa.  86400   IN      PTR     webserver.example.org.
```

### Knot DNS

Knot DNS je čistě autoritativní DNS server z díly CZ.NIC. Vyniká vysokým výkonem a podporou všech klíčových vlastností DNS. Knot je svobodný software šířený pod licencí GNU GPLv3 a funguje nejenom na Linuxu, ale také na BSD systémech.

Knot je pouze autoritativní server, což znamená, že umí pouze vracet data z vlastních zónových souborů, nikoliv provádět vyhledávání v DNS stromu (tj. nepracuje jako resolver).

Knot je součástí oficiálních repozitářů Debianu, a to počínaje vydáním Wheezy. Máte-li Debian Squeeze, budete muset buď server zkompilovat, nebo použít oficiální repozitář CZ.NIC, což realizujete úpravou/etc/apt/sources.list (popřípadě vytvořením nového souboru v adresáři /etc/apt/sources.list.d). Přidejte řádku s definicí repozitáře:

```
deb http://deb.knot-dns.cz/debian/ squeeze main
```

A následně aktualizujte místní databázi:

```
aptitude update
```

Poté, anebo pokud máte Debian Wheezy, můžete nainstalovat Knot obvyklým způsobem pomocí správce balíčků:

```
aptitude install knot
```

## Konfigurace Knotu

Základním konfiguračním souborem Knotu je /etc/knot/knot.conf, jehož obsah může v té nejjednodušší variantě vypadat takto:

```
system {
storage "/var/lib/knot";
}

interfaces {
lo-iface { address 127.0.0.1@53; }
#  all-ifaces { address 0.0.0.0@53; }
}

zones {
example.cz {
file "/etc/knot/example.cz.zone";
}
}
log {
syslog { any info, notice, warning, error; }
```

Proměnná *storage* ve skupině *system* udává, kam si bude Knot ukládat provozní data (více o těchto datech viz níže). V distribuci Knotu z repozitářů CZ.NIC pro Debian Squeeze tento adresář není založen, je tedy třeba jej vytvořit:

```
mkdir /var/lib/knot
```

Zpět ke konfiguračnímu souboru. Skupina *system* obsahuje některé další volby, jako například *workers*, pomocí kterých lze ručně nastavit počet vláken pro každý síťový interface.

Skupina *interfaces* definuje jednotlivá síťová rozhraní, kde Knot poběží. Jednotlivá rozhraní si můžete pojmenovat libovolně (tzn. *lo-iface* a *all-ifaces* můžete zaměnit za cokoliv, co je vám bližší). V příkladu jsou vidět dvě definovaná rozhraní, jedno pro lokální smyčku, druhé pro všechna IPv4 rozhraní. To druhé je zakomentované.

Následuje skupina *zones*, která obsahuje definici zón. V tomto případě je definována jediná zóna, a to*example.cz*, jejíž obsah bude uložen v souboru /etc/knot/example.cz.zone. Obsah tohoto souboru může vypadat třeba takto:

```
$TTL 2h
$ORIGIN example.cz.
@  IN  SOA  ns.example.cz. admin.example.cz. (
20120820 ; serial
4h       ; slave refresh
2h       ; slave retry interval
2w       ; slave data expiration
1h )     ; maximum caching time when lookups fail
)

NS    ns1.example.cz.
NS    ns2.example.cz.
MX    10 mail.example.cz.
MX    20 mail2.example.cz.

ns1           A     4.3.2.1
ns2           A     4.3.2.2

@             A     1.2.3.4
my            A     1.2.3.5
*.my          A     1.2.3.5
www           CNAME example.cz.
```

SOA záznam ani záznam ostatních snad již nemusím popisovat (minulý díl prakticky vše zmínil a vysvětlil).

Nastavení Knotu uvedené výše je osekané na kost – Knot umožňuje mnohem komplexnější nastavení, zejména v oblasti zónových souborů a transferu zón z primárních na sekundární DNS servery. Tato nastavení jsou i s příklady popsány v dokumentaci, kterou doporučuji si projít.

## Práce s Knotem

Samotné zónové soubory je třeba nejprve zkompilovat, a poté nastartovat Knot nebo provést jeho reload. Nejen k tomu slouží nástroj *knotc*. Důvodem kompilace je zrychlení startu, nevýhodou je pak na tuto nutnost pamatovat a po úpravách zónových souborů ji nezapomenout provést. Samotnou kompilaci můžete provést globálně pro všechny zóny:

```
knotc compile
```

Můžete být ale i selektivní (a třeba zkompilovat jen zóny *example.cz* a *example.org*):

```
knotc compile example.cz example.org
```

Knot také obsahuje nástroj pro kontrolu syntaxe konfiguračního souboru a pro kontrolu zónových souborů. Ty je vhodné provádět po úpravách ještě před kompilací, popřípadě před restartem nebo reloadem Knotu. Prověrku konfigurace provedete takto:

```
# knotc checkconf
2012-09-03T14:28:59.415223+02:00 Using '/etc/knot/knot.conf' as default configuration.
2012-09-03T14:28:59.415427+02:00 OK, configuration is valid.
```

K prověrce zónových souborů slouží příkaz:

```
knotc checkzone
```

Je možné kontrolovat jak globálně všechny zóny, tak pouze vybrané, a to stejným způsobem, jak je naznačeno u kompilace výše.

Samotný restart nebo reload Knotu je možné realizovat také pomocí nástroje *knotc*:

```
knotc reload
```

Funguje samozřejmě i restart:

```
knotc restart
```

## Správa linuxového serveru: Přístupová práva a ACL

Toto je poslední díl seriálu o správě linuxového serveru. Zatímco klasická unixová přístupová práva zná každý správce, o ACL a speciálních atributech už to tak obecně neplatí. Chcete vědět více? Čtěte dál.

### Unixová přístupová práva

Standardní unixová přístupová práva asi velmi dobře znáte. **Uživatel**, **skupina** a **ostatní** představují tři stěžejní subjekty, kterým se přidělují tři základní práva – právo **číst** (read), **psát** (write) a **spouštět**(execute). Chcete-li v unixových systémech spustit soubor obsahující spustitelný kód (binárka, shellový skript apod.), potřebujete souboru přidělit právo na spuštění.

V případě souborů je jasné, co jednotlivá práva povolují. V případě adresářů to už tak jasné být nemusí, nicméně v unixových systémech byste o adresáři měli uvažovat jako o speciálním typu souboru (v Unixech platí známé pravidlo *vše je soubor*). Pak už není těžké odvodit, co jednotlivá práva realizují:

- **read** – právo vypsat obsah adresáře (`ls`)
- **write** – právo upravovat obsah adresáře, tzn. vytvářet nové soubory a mazat existující (pozor: při mazání souborů se neberou v potaz přístupová práva k danému souboru, tzn. uživatel může vymazat soubor, ke kterému nemá žádná práva – stačí, pokud bude mít právo zápisu k adresáři, kde se soubor nachází)
- **execute** – právo se přepnout do daného adresáře (`cd`)

Aby to nebylo zase úplně jednoduché, existují u souborů i adresářů ještě tři speciální práva, Set UID, Set GID a sticky bit. SUID se užívá hlavně u binárek. Ty běží za normálních okolností s právy uživatele, který je spustil. Některé nástroje ale potřebují rootovská práva pro provedení určitých operací, a k tomu slouží SUID a SGID práva. SUID právo spustí daný program s právy jeho **vlastníka**, což je obvykle root. SGID právo spustí daný program s právy jeho **skupiny**. Co je velmi důležité zmínit – chyba v programech s těmito právy může vést k získání práv roota útočníkem, proto je třeba SUID/SGID binárky pečlivě hlídat a aktualizovat.

U adresářů má SGID bit specifický význam – nastavuje skupinu u všech souborů a adresářů vytvořených v tomto adresáři.

Sticky bit se dříve používal u binárek, kde signalizoval operačnímu systému, aby binárku po jejím ukončení neodstraňoval z operační paměti tak rychle. V dnešní době slouží u adresářů typu /tmp, kde zajišťuje, že soubory vytvořené jedním uživatelem nemůže smazat jiný uživatel, i když má k danému adresáři právo zápisu.

K nastavení vlastnictví slouží nástroj `chown`, na změnu práv pak `chmod`. Nástroj `chmod` můžete používat s více než jedním způsobem zápisu práv. Často se používá číselná reprezentace v osmičkové soustavě, kde právo číst má hodnotu **4**, právo zápisu hodnotu **2** a právo ke spuštění hodnotu **1**. Sečtením těchto čísel dostanete příslušnou hodnotu dané kombinace práv:

```
debian:/tmp# touch test
debian:/tmp# chown nobody:nogroup test
debian:/tmp# chmod 640 test
debian:/tmp# ls -l test
-rw-r----- 1 nobody nogroup 0 16. zář 15.15 test
```

Hodnota `640` dává vlastníkovi (6) právo číst a zapisovat (ale už ne spouštět), skupině (4) dává právo číst (ale už ne zapisovat a spouštět) a ostatním (0) nedává žádné právo.

Nástroj `chmod` také podporuje symbolický zápis, kde **u** představuje uživatele, **g** skupinu, **o** ostatní a **a**všechny tři skupiny. Následuje operátor, který přidává (**+**), odebírá (**-**) nebo nastavuje (**=**) práva. A konečně přicházejí na řadu práva:

- **r** – číst
- **w** – psát
- **x** – spouštět
- **s** – SGID / SUID
- **t** – sticky bit

Například:

```
chmod u=rw,g+s,o-r soubor
```

Tento příkaz nastavuje pro uživatele práva číst a zapisovat, ale ne spouštět, pro skupinu přidává Set GID bit (ostatní práva pro skupinu zachovává) a ostatním uživatelům bere právo číst.

### Atributy

Zajímavější jsou atributy, které lze vypisovat pomocí nástroje lsattr a nastavovat pomocí nástroje chattr. Těchto atributů je celá řada, ovšem musí je podporovat daný souborový systém. Nejzajímavější jsou dva, a sice append only (**a**) a immutable (**i**). První z nich povolí pouze zvětšení souboru, ale už ne modifikaci existujícího obsahu. Druhý pak zajistí, že soubor není možné jakkoliv měnit, upravovat nebo mazat, a to i když se jedná o uživatele root, pro kterého standardní práva nehrají roli. Přirozeně, máte-li roota, tak si příznak immutable můžete zrušit a soubor smazat. Přesto je dobré minimálně o těchto dvou atributech tušit. Nastavují se takto:

```
chattr +i immutable_file
chattr +a append_only_file
```

A ruší se takto:

```
chattr -i immutable_file
chattr -a append_only_file
```

Ostatní atributy najdete popsané v manuálové stránce nástroje chattr.

### ACL v Linuxu

Časem můžete zjistit, že klasická unixová přístupová práva nejsou dost jemná pro váš způsob použití. Chcete-li třeba dát konkrétnímu uživateli možnost číst nebo zapisovat do souboru, který vlastní někdo jiný a který je zařazen ve skupině, jejímž členem daný konkrétní uživatel není a nemá být, nedává vám unixový model přístupových práv žádný nástroj, jak toho dosáhnout. Zde jasně narážíte na jeho omezení.

V Linuxu však existují již poměrně dlouho ACL (Access Control List), které nabízejí jemnější model specifikace přístupových práv. Tuto vlastnost musí podporovat souborový systém a musíte explicitně stanovit příslušný atribut (**acl**) v /etc/fstab:

```
/dev/sda1 / ext3 defaults,errors=remount-ro,acl 0 0
```

Tím zajistíte připojení daného souborového systému s **acl**, ale až po restartu počítače. Za běhu systému souborový systém přenastavit můžete, a to i bez nutnosti jej odpojit a následně připojit, takto:

```
mount -o remount,acl /
```

ACL je třeba nějakým způsobem nastavit. Nástroje k tomuto účelu naleznete v případě Debianu v balíčkuacl:

```
aptitude install acl
```

Nástroj pro nastavení přístupových práv u souborů a adresářů se nazývá setfacl, nástroj pro vypsání stavu přístupových práv se jmenuje getfacl. Základní syntaxe ACL vypadá takto: **objekt**:**jméno**:**práva**, kde objekt je user pro uživatele (nebo zkráceně), group pro skupinu (nebo zkráceně g) a other pro ostatní (nebo zkráceně o). Jméno je jméno daného objektu (tedy daného uživatele nebo skupiny), v případě ostatních se (přirozeně) neuvádí. Práva představují klasickou trojici: číst (r), psát (w), spouštět (x).

Pro úpravu ACL se používá přepínač -m, pro mazání přepínač -x a pro vymazání všech ACL slouží přepínač -b.

Nejjednodušší použití představuje přidání sady práv nějakému uživateli nebo nějaké skupině:

```
debian:/home/staff# ls -l dokument.txt
-rw-r----- 1 michal staff 0 16. zář 15.12 dokument.txt
debian:/home/staff# setfacl -m u:honza:rw- dokument.txt
debian:/home/staff# getfacl dokument.txt
# file: dokument.txt
# owner: michal
# group: staff
user::rw-
user:honza:rw-
```

```
group::r--
mask::rw-
other::---
```

Není-li to z výpisu výše patrné, vězte, že ACL představuje rozšíření unixových práv. Unixová práva stále platí, ACL je pouze rozšiřuje. V příkladu výše tedy unixová práva stále umožňují členům skupiny staff číst daný soubor, vlastník (michal) má pak práva pro čtení a zápis, zatímco ostatní nemají žádná práva. V tomto příkladě byla pomocí ACL nastavena navíc práva uživateli honza, a to právo pro čtení a zápis do souborudokument.txt.

Práva se dají samozřejmě nastavit i skupině:

```
debian:/home/staff# setfacl -m g:www-data:r-- dokument.txt
debian:/home/staff# getfacl dokument.txt
# file: dokument.txt
# owner: michal
# group: staff
user::rw-
group::r--
group:www-data:r--
mask::rw-
other::---
```

V tomto příkladu bylo nastaveno právo pro čtení skupině www-data k souboru dokument.txt.

Pokud budete chtít sdílet nějaké materiály v nějaké skupině, bývá vhodné k tomu vyhradit celý adresář. Pokud však nastavíte pouze práva k danému adresáři a někdo v něm vytvoří soubor nebo další adresář, jak se v rámci ACL zajistí, aby i tyto nově vytvořené soubory a adresáře byly přístupné dané skupině nebo uživateli s danými ACL právy? Pomocí výchozích práv, pro jejichž nastavení slouží prefix **d:**, takto:

```
debian:/home# mkdir projekt
debian:/home# setfacl -m d:g:vyvojari:r-- projekt
debian:/home# cd projekt
debian:/home/projekt# touch dokument.txt
debian:/home/projekt# getfacl dokument.txt
# file: dokument.txt
# owner: root
# group: root
user::rw-
group::r-x                #effective:r--
group:vyvojari:r--
other::---
```

Zde je patrné, že nově vytvořený soubor dokument.txt v adresáři s patřičně nastavenými výchozími ACL zahrnuje přidané právo pro čtení skupině vyvojari.

ACL toho samozřejmě umí ještě o něco více, nicméně toto jsou základy, které je vhodné ovládat. Na zbytek se podívejte do manuálových stránek nástroje setfacl.

**Konec**

Tímto dílem bych se s vámi rád rozloučil, jelikož další díl již nevyjde. Včetně tohoto vyšlo celkem 98 dílů seriálu o správě linuxového serveru během téměř tří let. Když jsem začal psát první díly, neměl jsem sebemenší představu, že dosáhnu takového čísla nebo že vydržím psát tak dlouho. Dovolte mi vám poděkovat za přízeň, dále za všechny přívětivé a pochvalné komentáře, které mi dodávaly energii a chuť, a také za všechna doplnění a upřesnění.

Povedlo se mi pokrýt řadu témat a doufám, že jsem základy správy linuxových serverů dostatečně osvětlil. Určitě přijdete na řadu témat, kterou jsem nepokryl, ale to k věci patří. Správa linuxového serveru je velice široké téma, o kterém se toho dá napsat mnoho.

V zásadě, vlastní studium dokumentace a provádění rešerší je něco, co ke správě systémů patří. Žádný správce neví všechno. Stejně tak k tomu patří experimentování, testování a ladění, tedy praxe. To jsou má doporučení, kam jít dál.

Někteří z vás se ptali a ptají na to, jestli by bylo možné získat celý seriál v nějakém rozumném celku k vytištění nebo k prohlížení na cestách. V tomto ohledu mám pro vás dobrou zprávu – připravujeme vydání seriálu v knižní podobě v edici CZ.NIC.

Tímto se s vámi loučím a přeji vám hodně štěstí a úspěchů (nejen) v oblasti správy linuxových serverů.