Trials, Tribulations & VHost Misconfigurations

...

Samuel Erb -- VirSecCon2020 5/4/2020

whoami

Samuel Erb @erbbysam

Software Engineer

DEF CON 23, 24 Black Badge (Badge Challenge)

HackerOne h1-415 2019 “Most Valuable Hacker”

(disclaimer)

The views expressed here are my own.

Nothing presented here gives you permission to hack. Always seek explicit approval.

A developer is presented with a problem

They want to host 3 websites:

1) **https://www.example.com**
2) **https://blog.example.com**
3) **https://internal-dev.example.com**

A developer is presented with a problem

They want to host 3 websites:

1) https://www.example.com
2) https://blog.example.com
3) https://internal-dev.example.com

Server A
Runs Apache
Listens on port 443
hosts www.example.com

Server B
Runs Apache
Listens on port 443
hosts blog.example.com

Server C
Runs Apache
Listens on port 443
hosts internal-dev.example.com

Perfectly fine solution! Solution won't scale to an arbitrary # of subdomains, but that's probably ok.

A developer is presented with a problem

They want to host 3 websites:

1) https://www.example.com
2) https://blog.example.com
3) https://internal-dev.example.com

Server A

Runs Apache
Listens on port 443
hosts www.example.com

Runs Apache
Listens on port 444
hosts blog.example.com

Runs Apache
Listens on port 445
hosts internal-dev.example.com

X -- asking a user to type in https://blog.example.com:444 is non-standard!

A website owner is presented with a problem

They want to host 3 websites:

1) https://www.example.com
2) https://blog.example.com
3) https://internal-dev.example.com

Server A

Runs Apache
Listens on port 443
Hosts all 3 sites using a VHost Configuration

Works great, but what's a VHost?

Apache VHost Configuration File

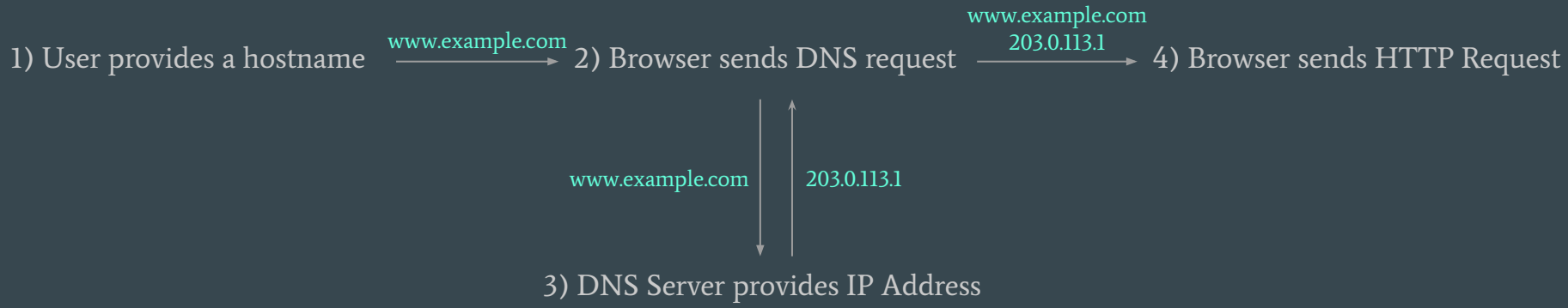
```
LoadModule ssl_module modules/mod_ssl.so
Listen 443
<VirtualHost *:443>
      DocumentRoot "/dev-internal/"
    ServerName dev-internal.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/dev-internal.example.com.cert"
    SSLCertificateKeyFile "/secrets/dev-internal.example.com.key"
</VirtualHost>
<VirtualHost *:443>
      DocumentRoot "/www/"
    ServerName www.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/www.example.com.cert"
    SSLCertificateKeyFile "/secrets/www.example.com.key"
</VirtualHost>
<VirtualHost *:443>
      DocumentRoot "/blog/"
    ServerName blog.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/blog.example.com.cert"
    SSLCertificateKeyFile "/secrets/blog.example.com.key"
</VirtualHost>
```

Server A

Runs Apache
Listens on port 443
Hosts all 3 sites using a VHost Configuration

Works great, but what's a VHost?

Before we go any further, a quick networking break



Note:
example.com is reserved for documentation use by https://tools.ietf.org/html/rfc2606
203.0.113.1 is reserved for documentation use by https://tools.ietf.org/search/rfc5737

Before we go any further, a quick networking break

Networking Layer	Protocol	
5 -- Application	HTTP 1.1	www.example.com
4 -- Presentation	TLS 1.2	www.example.com
3 -- Transport	TCP	
2 -- Network	Network / IPv4	203.0.113.1
1 -- Physical	Wire protocol	

TLS Handshake

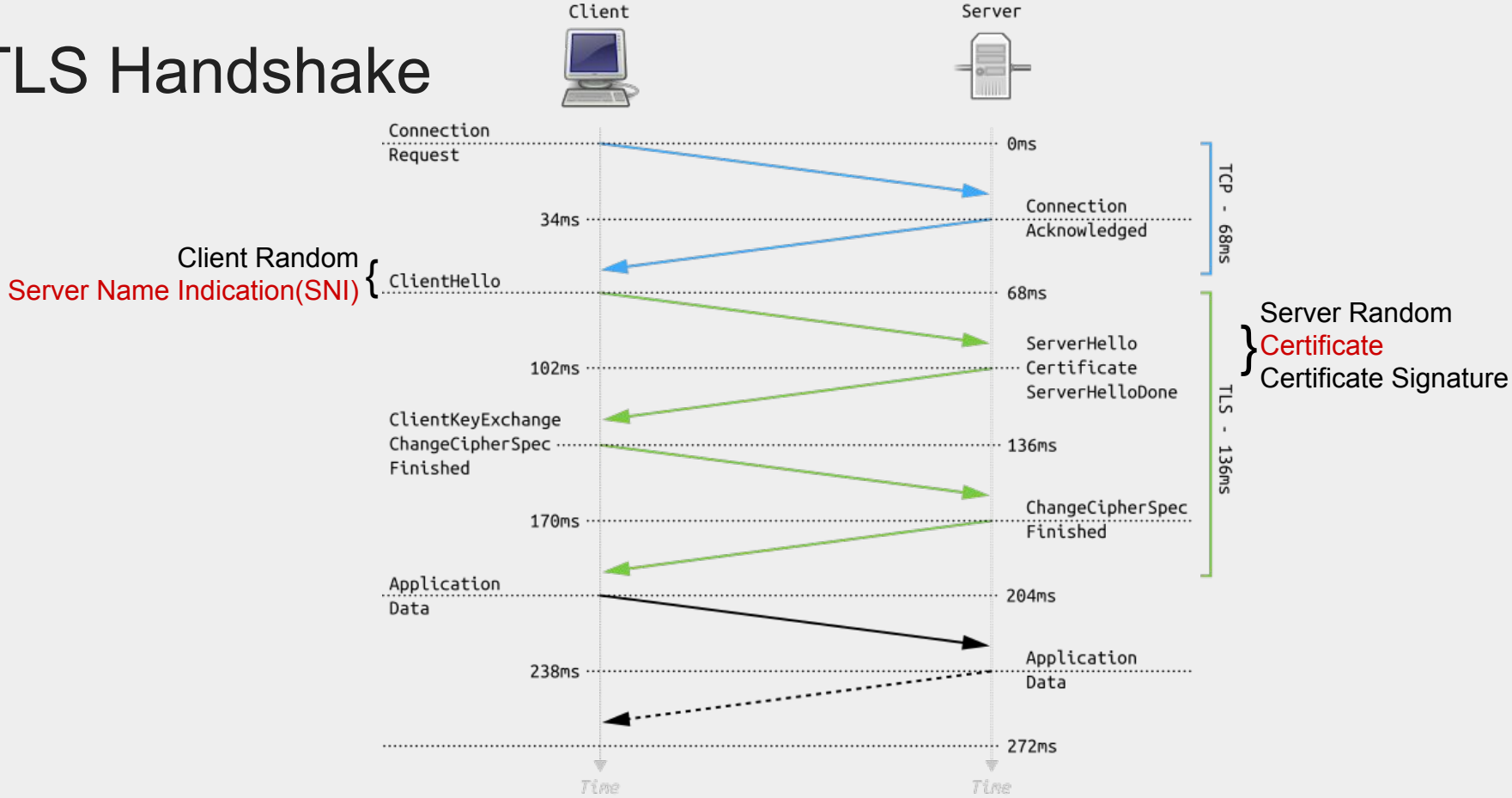
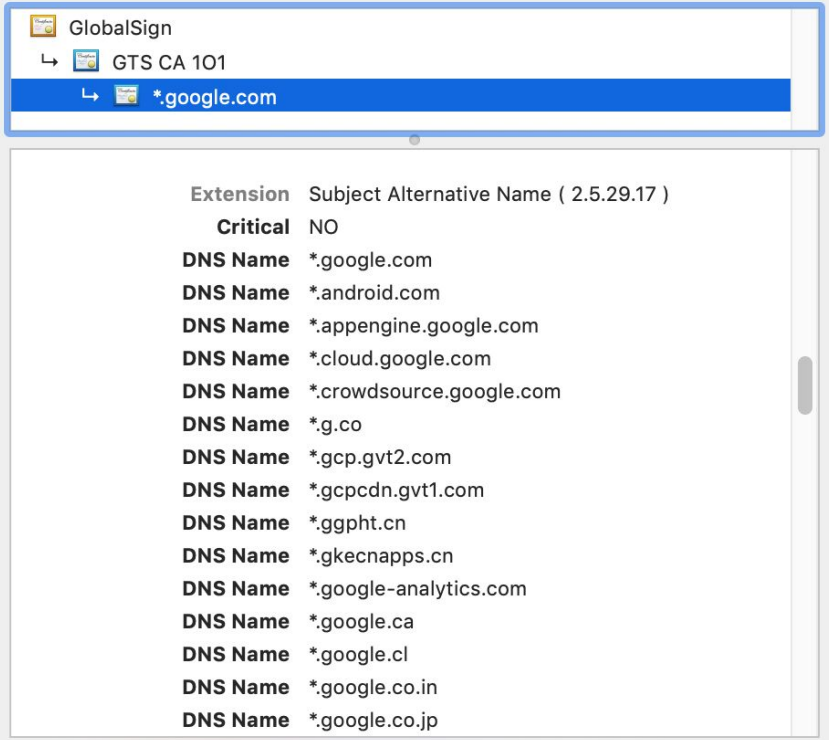


Image via https://commons.wikimedia.org/wiki/File:Full_TLS_1.2_Handshake.svg

TLS Handshake

```
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
  ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 508
      Version: TLS 1.2 (0x0303)
    ▶ Random
      Session ID Length: 32
      Session ID: 3c17fe3c9cb51700745aad450d843b61e915530b74fc4082...
      Cipher Suites Length: 34
    ▶ Cipher Suites (17 suites)
      Compression Methods Length: 1
    ▶ Compression Methods (1 method)
      Extensions Length: 401
    ▶ Extension: Unknown 39578
    ▼ Extension: server_name
        Type: server_name (0x0000)
        Length: 16
      ▼ Server Name Indication extension
          Server Name list length: 14
          Server Name Type: host_name (0)
          Server Name length: 11
          Server Name: example.com
```



HTTP Request

GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1

Historical protocols?

HTTP 0.9 (~1991):

```
GET /index.html HTTP/0.9
```

HTTP 1.1 (~1996):

```
GET /my-page.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

HTTP 1.1 (~1997):

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X ...
Accept: text/html,application/xhtml+xml,...
...
```

TLS 1.1 (~2006) client hello without extensions:

```
struct {
      ProtocolVersion client_version;
      Random random;
      SessionID session_id;
      CipherSuite cipher_suites<2..2^16-1>;
      CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;
```

TLS 1.1 (~2006) extended client hello with extensions:

```
struct {
        ProtocolVersion client_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
        CompressionMethod compression_methods<1..2^8-1>;
        Extension client_hello_extension_list<0..2^16-1>;
} ClientHello;
```

One possible extension is Server Name Indication (SNI)

ref: https://medium.com/platform-engineer/evolution-of-http-69cfe6531ba0
ref: https://tools.ietf.org/html/rfc4366
ref: https://tools.ietf.org/html/rfc4346

OK, back to VHosts

A client will send enough information to differentiate hosts on a single server:

- TLS Server Name Indicator (SNI)
- HTTP “Host” header

This allows multiple virtual hosts to reside at the same IP address.

OK, back to VHosts

A client will send enough information to differentiate hosts on a single server:

- TLS Server Name Indicator (SNI)
- HTTP “Host” header

This allows multiple virtual hosts to reside at the same IP address.

But what if the client doesn’t send either?

What if no TLS SNI or Host header are sent?

Websites needed a default until very recently. Even a 1% user failure rate is significant.

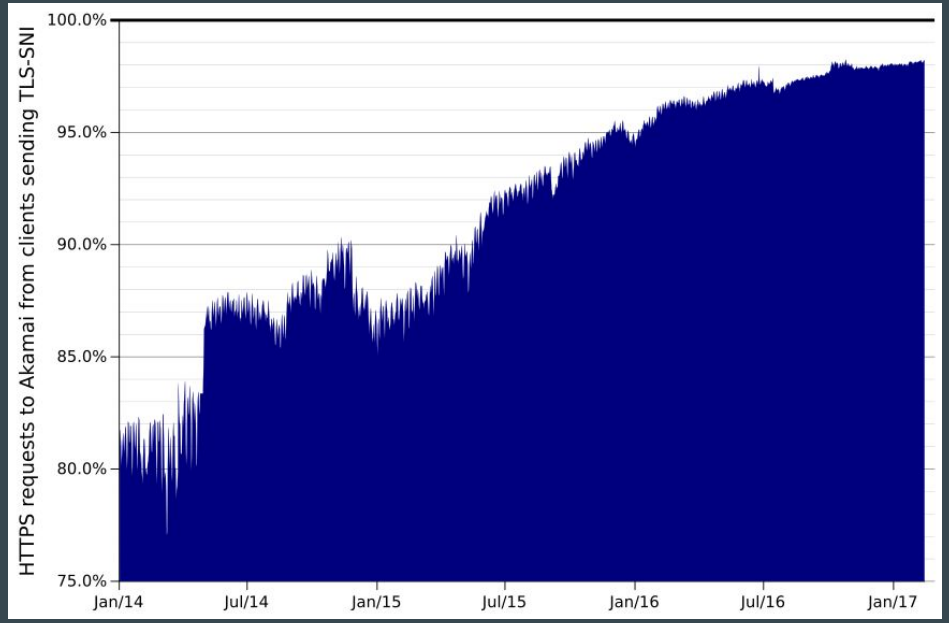


Image source: https://blogs.akamai.com/2017/03/reaching-toward-universal-tls-sni.html

This default behavior might not be what you expect...

```
LoadModule ssl_module modules/mod_ssl.so
Listen 443
<VirtualHost *:443>
      DocumentRoot "/dev-internal/"
    ServerName dev-internal.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/dev-internal.example.com.cert"
    SSLCertificateKeyFile "/secrets/dev-internal.example.com.key"
</VirtualHost>
<VirtualHost *:443>
      DocumentRoot "/www/"
    ServerName www.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/www.example.com.cert"
    SSLCertificateKeyFile "/secrets/www.example.com.key"
</VirtualHost>
<VirtualHost *:443>
      DocumentRoot "/blog/"
    ServerName blog.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/blog.example.com.cert"
    SSLCertificateKeyFile "/secrets/blog.example.com.key"
</VirtualHost>
```

Server A

Runs Apache
Listens on port 443
Hosts all 3 sites using a VHost Configuration

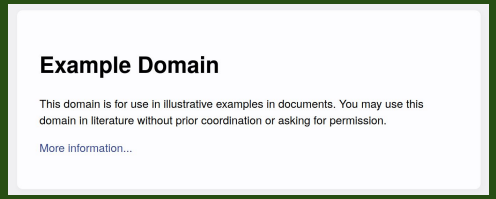
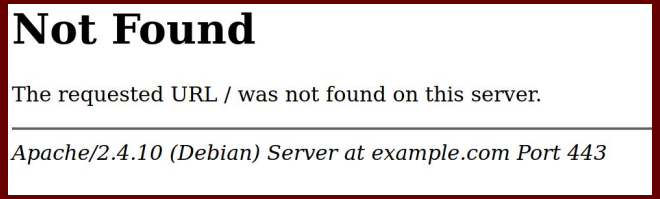
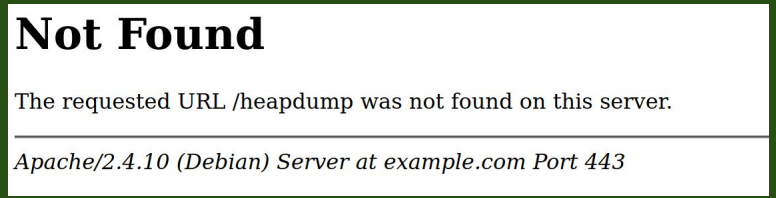
Fixing the Problem

```
LoadModule ssl_module modules/mod_ssl.so
Listen 443
<VirtualHost *:443>
      DocumentRoot "/www/"
    ServerName www.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/www.example.com.cert"
    SSLCertificateKeyFile "/secrets/www.example.com.key"
</VirtualHost>
<VirtualHost *:443>
      DocumentRoot "/dev-internal/"
    ServerName dev-internal.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/dev-internal.example.com.cert"
    SSLCertificateKeyFile "/secrets/dev-internal.example.com.key"
</VirtualHost>
<VirtualHost *:443>
      DocumentRoot "/blog/"
    ServerName blog.example.com
    SSLEngine on
    SSLCertificateFile "/secrets/blog.example.com.cert"
    SSLCertificateKeyFile "/secrets/blog.example.com.key"
</VirtualHost>
```

Server A

Runs Apache
Listens on port 443
Hosts all 3 sites using a VHost Configuration

Case Study -- So what's the worst that can happen?

Host header, TLS SNI	No Host header / TLS SNI
GET / HTTP/1.1 Host: www.example.com  	GET / HTTP/1.1 
GET /heapdump HTTP/1.1 Host: www.example.com  	GET /heapdump HTTP/1.1 

Case Study -- So what's the worst that can happen? (part 2)

Hostname	DNS Lookup Result
www.example.com	203.0.113.7
blog.example.com	203.0.113.7

GET /admin/execute HTTP/1.1
Host: blog.example.com

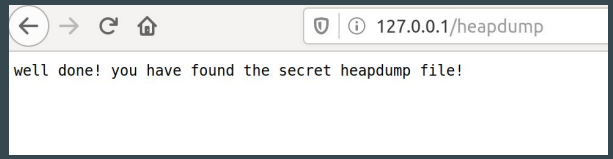
Live Example

Simple docker container, runs locally:

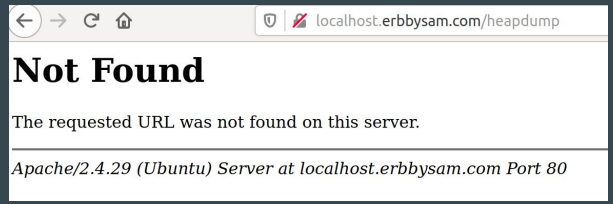
```
<VirtualHost *:80>
  ServerName invalid-dev.erbbysam.com
  DocumentRoot /var/www/site/dev
</VirtualHost>

<VirtualHost *:80>
  ServerName localhost.erbbysam.com
  DocumentRoot /var/www/site/www
</VirtualHost>
```

127.0.0.1/heapdump →



localhost.erbbysam.com/heapdump →



Try it yourself: https://github.com/erbbysam/docker-vuln-vhosts

Red Team Ideas

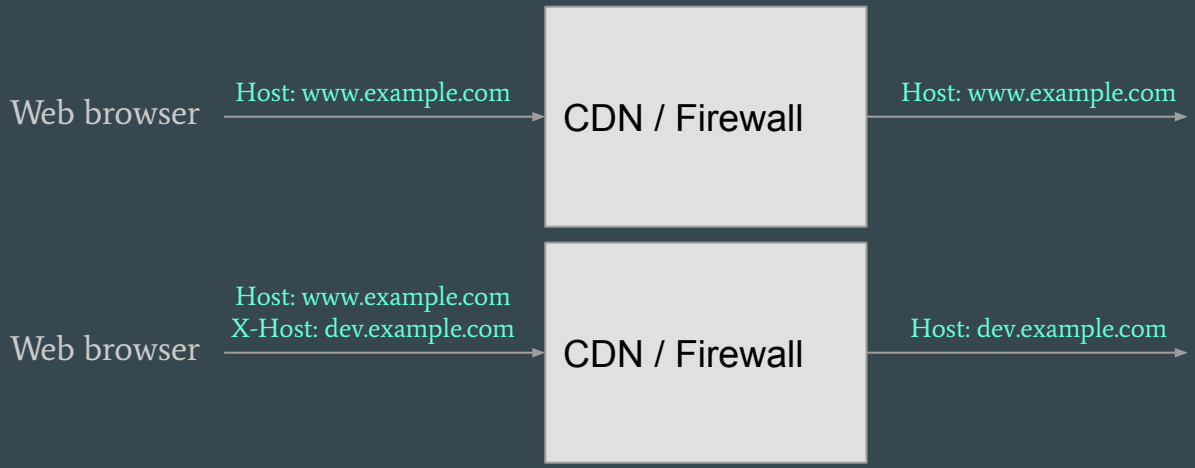
- Look for multiple hosts resolving to the same IP
- “Vhost Scanners” exist to assist identifying this scenario
  - https://github.com/codingo/VHostScan
  - https://github.com/jobertabma/virtual-host-discovery
  - Be careful to avoid impacting the performance of a server
- Attempt to guess paths using simple wordlist without a host header!
  - Keep it short, and use it on *every* host you encounter
  - Use meg! https://github.com/tomnomnom/meg
  - meg has a useful wordlist for this: https://github.com/tomnomnom/meg/blob/master/lists/configfiles

Blue Team Ideas

YOU ARE AT AN INFORMATION ADVANTAGE!

- Run everything the on the previous slide, but skip the brute forcing
  - Use internal DNS information
  - Parse internal configuration files
- Investigate default behavior -- until very recently (~2016+) developers needed to support missing TLS SNI

Firewall / CDN & Host Headers



- Read documentation, look for old defects & common misconfigurations.
- “X-Host” here is used strictly as an example above to override the user supplied Host header in the forwarded request.

Case Study -- TALOS-2018-0702

A firewall would filter malicious Host headers. It could be bypassed using a large packet.

```
- Packet 1
GET
/0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000

- Packet 2
000000000000000000000000000000000000000000000000000000000 HTTP/1.1
Host: ${MALICIOUS_HOST}
User-Agent: curl/7.61.1


   Accept: */*
```

Ref: https://talosintelligence.com/vulnerability_reports/TALOS-2018-0702
Discovered by Claudio Bozzato of Cisco Talos

Case Study -- Host of Troubles

What happens if you send the following request?

```
GET http://victim.com/ HTTP/1.1
Host: attack.com
```

victim.com could be cached with the IP address of attack.com :(

Widely fixed in CDNs, firewalls & proxies in 2016.

Ref: https://hostoftroubles.com/
Ref: http://www.icir.org/vern/papers/host-of-troubles.ccs16.pdf
Ref: https://tools.ietf.org/html/rfc7230#section-5.3.2

Takeaways

- Look for vhosts
  - Red team: brute force
  - Blue team: use available internal information
- Know the architecture & know the defaults
- Always, always try direct IP access (no Host header / TLS SNI)

Takeaways

- Look for vhosts
  - Red team: brute force
  - Blue team: use available internal information
- Know the architecture & know the defaults
- Always, always try direct IP access (no Host header / TLS SNI)

Thank you. Stay safe <3

Questions, comments -- @erbbysam