Bypassing HTTP Strict Transport Security

Jose Selvi
Twitter: @JoseSelvi
Jose Selvi

+10 years working in security

Principal Penetration Tester

SANS Institute Community Instructor

GIAC Security Expert (GSE)

Twitter: @JoseSelvi

Blog: http://www.pentester.es
Not a Silver Bullet
Let’s Go!

- History of Bypassing SSL
- HTTP Strict Transport Security
- HSTS Weakness
  - *****************
  - ***************
  - *******************
False SSL Certificate

Client → HTTPS → Attacker → HTTPS → Server

This Connection is Untrusted
You have asked Firefox to connect securely to [redacted], but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

- Get me out of here!
- Technical Details
- I Understand the Risks
PKI Compromise
Design weaknesses

- BEAST / CRIME
  - By Juliano Rizzo & Thai Duong
- BREACH
  - By Angel Prado, Neal Harris & Yoel Gluck
- Based on compression characteristics before encryption
- Chosen plaintext attack
- It can decrypt secrets (cookie, csrf-token, etc)
Implementation weaknesses

```c
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
uint8_t *signature, UInt16 signatureLen)
{
    OSStatus err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
            goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

    fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```
Stripping SSL Links

GET / HTTP/1.1

<body>
<img src=whatever.jpg>
<a href = https://myweb/login>
</body>
Let’s Go!

• History of Bypassing SSL
• HTTP Strict Transport Security
• HSTS Weakness

************************
• **************
• ************************
• ***************
• **************************
HTTP Strict Transport Security

- RFC-6797: Published in November 2012.
- Also known as HSTS or STS.
- Prevent HTTP connections.
- Prevent accepting self-signed and rogue certificates.
- Use a new “Strict-Transport-Security” header.
Who uses HSTS?

http://paul.vanbrouwershaven.com/2014/05/everyone-needs-http-strict-transport.html
Who uses HSTS?

- LogMeIn
- Dropbox
- GitHub
- Facebook
- Google
- PayPal
- Microsoft
- Yahoo!
Browsers support

http://caniuse.com/#feat=stricttransportsecurity
HTTPS Strict Transport Security

Client

HTTPS

Server

GET / HTTP/1.1

Strict-Transport-Security: max-age=3153600
HTTP Strict Transport Security

- **max-age**: number of seconds that the policy is enabled.

  max-age=0 -> Delete policy

- **includeSubdomains**: If present, the policy applies all subdomains, not just the visited one.

```
$ ./hsts_catcher.py -U https://accounts.google.com
max-age=10893354; includeSubDomains
$

$ ./hsts_catcher.py -U https://paypal.com
max-age=14400
$

$ ./hsts_catcher.py -U https://github.com
max-age=31536000; includeSubDomains; preload
```
HSTS Timeline

HTTPS connection

3153600 secs later
Preloaded HSTS

- Harcoded list of well known website names that should use always HTTPS.
- Prevent the security gap before the first HTTPS connection.
- Google, Twitter, Paypal, ...
Let’s Go!

• History of Bypassing SSL
• HTTP Strict Transport Security
• HSTS Weakness

• ************************
• ****************************
• ***********************
• ******************************
Too short max-age


HTTP/1.1 301 Moved Permanently
Server: Apache
Cache-Control: private
Pragma: no-cache
Expires: Thu, 05 Jan 1995 22:00:00 GMT
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=14400
Location: https://www.paypal.com/es/webapps/mpp/home
Strict-Transport-Security: max-age=14400
Content-Type: text/html; charset=UTF-8
DC: slc-a-origin-www-2.paypal.com
Date: Sun, 14 Sep 2014 15:26:01 GMT
Connection: keep-alive
Vary: Accept-Encoding
Connection: Transfer-Encoding

4 hours
Looking for weaknesses
HSTS Timeline
HTTPS connection
3153600 secs later

3153600 secs later
Preloaded HSTS - Google

There is still a window where a user who has a fresh install, or who wipes out their local state, is vulnerable. Because of that, Chrome and Firefox share a "Preloaded HSTS" list. These domains will be configured for HSTS out of the box.

If you own a site that you would like to see included in the preloaded HSTS list you can submit it at https://hstspreload.appspot.com.

A selected subset of the members of the preloaded HSTS list:

- Google
- Paypal
- Twitter
- Simple
- Linode
- Stripe
- Lastpass

Check the source for the full list.

http://www.chromium.org/sts
However, when connecting to an HSTS host for the first time, the browser won’t know whether or not to use a secure connection, because it has never received an HSTS header from that host. Consequently, an active network attacker could prevent the browser from ever connecting securely (and even worse, the user may never realize something is amiss). To mitigate this attack, we have added to Firefox a list of hosts that want HSTS enforced by default. When a user connects to one of these hosts for the first time, the browser will know that it must use a secure connection. If a network attacker prevents secure connections to the server, the browser will not attempt to connect over an insecure protocol, thus maintaining the user’s security.
void NetInternalsMessageHandler::IOThreadImpl::OnHSTSAdd(
  const base::ListValue* list) {
  // |list| should be: [<domain to query>, <STS include subdomains>, <PKP
  // include subdomains>, <key pairs>]
  std::vector<std::string> strings;
  CHECK(list->GetList(strings));
  if (!strings.empty()) {
    if (!transport_security_state_)
      return;

    base::Time expiry = base::Time::Now() + base::TimeDelta::FromDays(1000);
    net::TransportSecurityState* transport_security_state =
        GetMainContext()->transport_security_state();
    net::HashValueVector hashes;
    if (!hashes_str.empty()) {
      if (!Base64StringToHashes(hashes_str, &hashes))
        return;

    transport_security_state_->AddHSTS(domain, expiry, sts_include_subdomains);
    transport_security_state_->AddHPKP(domain, expiry, pkp_include_subdomains,

Safari PList

$ plutil -p HSTS.plist
{
  "com.apple.CFNetwork.defaultStorageSession" => {
    "ssl.google-analytics.com" => -inf
    "webmail.mayfirst.org" => -inf
    "braintreegateway.com" => -inf
    "code.google.com" => -inf
    "dm.mylookout.com" => inf
    "therapynotes.com" => inf
    "chrome.google.com" => -inf
    "sol.io" => -inf
    "www.sandbox.mydigipass.com" => inf
  }
[...]
HSTS Weakness

• Its security relies on time.
• It completely trust the OS’s current time.
• Is it trustable?
• Is it possible to change the system time from the network?
Let’s Go!

- History of Bypassing SSL
- HTTP Strict Transport Security
- HSTS Weakness
- Network Time Protocol (NTP)
- Get in a Delorean
- OS Time Synchronisation & Browsers
Network Time Protocol (NTP)

- Time Synchronisation Services.
- RFC-1305 (v3) / RFC-5905 (v4) / RFC-4330 (SNTPv4).
- Set up by default on most (or all) Operating Systems.
- Security features (v4) NOT used by default.
- Vulnerable to Man-in-the-Middle techniques.
Network Time Protocol (NTP)

It's 11:00

Actually It's 11:02
<table>
<thead>
<tr>
<th>LI</th>
<th>VN</th>
<th>Mode</th>
<th>Stratum</th>
<th>Poll</th>
<th>Precisión</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Root Delay</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Root Dispersion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reference Identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reference Timestamp (64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Originate Timestamp (64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Receive Timestamp (64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transmit Timestamp (64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key Identifier (optional) (32)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Message Digest (optional) (128)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NTP Packet (II)

- Leap: 3 -> Clock not synchronised.
- Version: 3 /4
- Mode: Client (3) / Server (4) / etc.
- Poll: NTP polling interval.
- Precision: Usually -20 ($2^{-20}$) -> Microseconds.
NTP Packet (III)

- Root delay & dispersion: NTP short format.
- Timestamps: NTP timestamp format
Example: Ubuntu Linux

Network Time Protocol (NTP Version 4, client)

Flags: 0xe3

11... = Leap Indicator: unknown (clock unsynchronized) (3)
..10 0... = Version number
.... .011 = Mode: client
Peer Clock Stratum: unspecified
Peer Polling Interval: invalid
Peer Clock Precision: 0.011
Root Delay: 1.00000 sec
Root Dispersion: 1.00000
Reference ID: NULL
Reference Timestamp: Jan 1, 1970
Receive Timestamp: Jan 1, 1970
Transmit Timestamp: Jan 1, 1970

Network Time Protocol (NTP Version 4, server)

Flags: 0x24

00... = Leap Indicator: no warning (0)
..10 0... = Version number: NTP Version 4 (4)
.... .100 = Mode: server (4)
Peer Clock Stratum: secondary reference (2)
Peer Polling Interval: invalid (3)
Peer Clock Precision: 0.000001 sec
Root Delay: 0.0099 sec
Root Dispersion: 0.0239 sec
Reference ID: 192.93.2.20
Reference Timestamp: Sep 3, 2014 08:36:01.601928000 UTC
Origin Timestamp: Sep 3, 2014 08:40:04.634295000 UTC
Receive Timestamp: Sep 3, 2014 08:40:04.653302000 UTC
Transmit Timestamp: Sep 3, 2014 08:40:04.653354000 UTC
NTP Man-in-the-Middle

Oct 21 2015 07:28

It’s 11:00

Victim

Actually It’s

Oct 21 2015 07:28

Fake NTP

NTP

blackhat
EUROPE 2014
Let’s Go!

- History of Bypassing SSL
- HTTP Strict Transport Security
- HSTS Weakness
- Network Time Protocol (NTP)
- Get in a Delorean
- OS Time Synchronisation & Browsers
Delorean

  - [http://github.com/PentesterES/Delorean](http://github.com/PentesterES/Delorean)
- Inspired on a kimifly’s work:
  - [http://github.com/limifly/ntpserver](http://github.com/limifly/ntpserver)
- Implements some attacks.
- Pretend to become an NTP attack suite.
$ ./delorean.py -h
Usage: delorean.py [options]

Options:
- h, --help show this help message and exit
- i INTERFACE, --interface=INTERFACE Listening interface
- p PORT, --port=PORT Listening port
- n, --nobanner Not show Delorean banner
- s STEP, --force-step=STEP Force the time step: 3m (minutes), 4d (days), 1M (month)
- d DATE, --force-date=DATE Force the date: YYYY-MM-DD hh:mm[:ss]
- r, --random-date Use random date each time
Let’s Go!

- History of Bypassing SSL
- HTTP Strict Transport Security
- HSTS Weakness
- Network Time Protocol (NTP)
- Get in a Delorean
- OS Time Synchronisation & Browsers
Ubuntu Linux

• Really simple.
• NTPv4. No authentication.
• At each network reconnection (& boot time).

$ ls /etc/network/if-up.d/
000resolvconf  avahi-daemon  ntpdate  wpasupplicant
avahi-autoipd  ethtool  upstart
Fedora Linux

• The simplest one.
• NTPv3.
• More than one NTP server.
• EACH minute!

$ tcpdump -i eth0 -nn src port 123
12:44:55.696390 IP 192.168.1.101.123 > 213.194.159.3.123: NTPv3, Client, length 48
12:45:59.034059 IP 192.168.1.101.123 > 89.248.106.98.123: NTPv3, Client, length 48
Mac OS X - Lion

- Pretty simple as well.
- NTPv4. No authentication.
- Each 9 minutes.

$ tcpdump -i eth0 -nn src port 123
09:02:18.166708 IP 192.168.1.100.123 > 17.72.148.53.123: NTPv4, Client, length 48
09:11:20.059792 IP 192.168.1.100.123 > 17.72.148.53.123: NTPv4, Client, length 48
09:20:17.951361 IP 192.168.1.100.123 > 17.72.148.53.123: NTPv4, Client, length 48
Mac OS X - Mavericks

- New synchronisation service.
- NTP still exists but not synchronises.
  - Just write in /var/db/ntp.drift
- A new service called “pacemaker” check this file and synchronise the system clock.
- It seems it doesn’t work as expected...

Mac OS X - Mavericks
• NTPv3 but...
• The securest one.
• Synchronization each 7 days.
• Doesn’t accept more than 15 hours increment/decrement.
• Domain members have a different set up.
**W32time Service**

```
<Task>
  <MaintenanceSettings>
    <Period>P7D</Period>
    <Deadline>P14D</Deadline>
    <Exclusive>false</Exclusive>
  </MaintenanceSettings>
  <WakeToRun>false</WakeToRun>
  <ExecutionTimeLimit>P3D</ExecutionTimeLimit>
  <Priority>7</Priority>
</Settings>

  <Actions Context="LocalService">
    <Exec>
      <Command>%windir%\system32\sc.exe</Command>
      <Arguments>start w32time task_started</Arguments>
    </Exec>
  </Actions>
</Task>
```
Max[Pos | Neg]PhaseCorrection

W7 / W8
15 hours

W2K12 48 hours
Time Skimming Attack

Time Sync

153600 secs later
Force Synchronisation

This computer is set to automatically synchronize with 'time.windows.com'.

Next synchronization: 8/31/2014 at 1:00 AM

An error occurred while Windows was synchronizing with time.windows.com. This operation returned because the timeout period expired.

The clock was most recently synchronized on 8/29/2014 at 1:00 AM.
Let’s Go!

- History of Bypassing SSL
- HTTP Strict Transport Security
- HSTS Weakness
- Network Time Protocol (NTP)
- Get in a Delorean
- OS Time Synchronisation & Browsers
Whose fault is?
Thanks for the report, but this is a fairly well known, generic issue with relying on unauthenticated NTP and is not specific to HSTS. Consider that SSL/TLS, Kerberos, and AD Active Directory all have the same underlying concerns and attack profile. The typical solutions for this are to use authenticated NTP, use multiple NTP servers with a client that detects excessive drift or "bad tickers", or to use some other mechanism to provide authenticated time.
References

- https://www.owasp.org/index.php/HTTP_Strict_Transport_Security
- http://dev.chromium.org/sts
- http://www.ntp.org
- https://github.com/limifly/ntpserver
- http://www.thoughtcrime.org/software/sslstrip/
- https://github.com/LeonardoNve/dns2proxy
Thanks! Questions?

Jose Selvi

http://twitter.com/JoseSelvi

jselvi@incide.es
http://www.incide.es

jselvi@pentester.es
http://www.pentester.es