Breaking SSL using time synchronisation attacks

Jose Selvi, Senior Security Consultant
$ whois jselvi

- Jose Selvi
- +10 years working in security
- Senior security Consultant
- SANS Institute Community Instructor
- GIAC Security Expert (GSE)
- Twitter: @JoseSelvi
- Blog: http://www.pentester.es
Warning! Spanish accent!
Let’s Go!

• Modern Time Synchronisation
• Get in a Delorean
• HTTP Strict Transport Security
• Windows task scheduler
• Public Key Infrastructure
• Conclusions & Recommendations
Network Time Protocol (NTP)

- Time Synchronisation Services.
- RFC-1305 (v3) / RFC-5905 (v4) / RFC-4330 (SNTPv4).

- By default in (almost) all operating systems.

- No secured by default.
- Vulnerable to Man-in-the-Middle attacks.
### NTP Packet

<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></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Root Delay**
- **Root Dispersion**
- **Reference Identifier**
- **Reference Timestamp (64)**
- **Originate Timestamp (64)**
- **Receive Timestamp (64)**
- **Transmit Timestamp (64)**
- **Key Identifier (optional) (32)**
- **Message Digest (optional) (128)**
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: 0.010
  - Root Delay: 1.00000 sec
  - Root Dispersion: 1.0000
  - Reference ID: NULL
  - Reference Timestamp: Jan 1, 1970
  - Origin 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
Mac OS X - Mavericks

- New synchronisation service
- NTP daemon exits, but not synchronises.
- Just writes in `/var/db/ntp.drift`
- A new service called “pacemaker” check that file and change the clock.
- It seems it doesn’t work as it should...

Does NTP work?

N7RJN
Does not accurately do so.

Does anyone have a suggested solution to this issue?

upland_rage
Nov 26, 2013 10:41 AM

Can not keep time sync'd. I rely on time stamping and can see time drift from being seconds to being minutes behind. When I run ntpq -np poll interval shows 64 but "when" maybe several thousand since it polled last. I have also tried different time servers. This only started with the upgrade to Mavericks.

MacBook Air, OS X Mavericks (10.9)

This solved my question by upland_rage on Dec 3, 2013 8:23 AM

I compiled the latest version of NTP from NTP.org and it has been working perfectly all weekend.

See the answer in context
LOG=/var/run/sntp.log

ipconfig waitall

if [[ ! -f ${LOG} ]]; then
DEADLINE=${((SECONDS+TIMEOUT))}
for (( CURTIMEOUT=TIMEOUT; SECONDS < DEADLINE; CURTIMEOUT=DEADLINE-SECONDS )); do
if scutil -w ${KEY} -t ${CURTIMEOUT}; then
if [[ -f ${DNS} ]]; then
break;
fi
    # else retry false alarms
else
logger -p daemon.err "$0: scutil key ${KEY} not present after ${TIMEOUT} seconds"
break;
fi
done
fi

for server in $(awk '/^server/ {print $2}' /etc/ntp.conf); do
    sntp -K /dev/null -s ${server} &> ${LOG};
if then
break
else
logger -p daemon.err -f ${LOG}
fi
done

exec /usr/sbin/ntpd -c /private/etc/ntp-restrict.conf -d -D 10 -n -g -p /var/run/ntpd.pid -f /var/db/ntp.drift
Mac OS X - Mavericks
Fedora Linux

- The easiest
- NTPv3.
- More than one NTP server
- Requests each minute!

```bash
$ 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
```
Ubuntu Linux

• Very simple
• NTPv4.
• Each time it connects to a network (and at boot time, of course).

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

- NTPv3 but...
- The most secure.
- Synchronisation each 7 days.
- More than 15 hours drift isn’t allowed.
- Domain members work in a different way.
W32time service

```xml
<Settings>
  <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>
```

And servers in the network. If this service is stopped, the service is disabled, any services that explicitly
Max[Pos|Neg]PhaseCorrection

W7 / W8
15 horas

W2K12 48 horas
What the Internet says?

Force Windows time synchronization more often

Last week we discussed how to setup synchronization with an external time source. This week we will learn how to make the syncs

Automatically sync windows time more often than default

I have a few PCs that are losing time, and I'd like windows to synch them more often with the internet time. I think the windows default attempts to update only once per day, and does not update if the time server is not available (which seems to happen quite often) meaning the PCs can end up 20 or 30 seconds out.

I'd like to create a scheduled task to do this say every 5 mins, and if the default time server is not available use mut
Manual Synchronisation

This computer is set to automatically synchronize its time 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.
Network Time Protocol (NTP Version 3, server)

Flags: 0x1c
Peer Clock Stratum: primary reference (1)
Peer Polling Interval: 17 (131072 sec)
Peer Clock Precision: 0.015625 sec
Root Delay: 0.0000 sec
Root Dispersion: 10.8970 sec
Reference ID: uncalibrated local clock
Transmit Timestamp: Oct  7, 2014 08:28:38.118040000 UTC

Key ID: 5e040000
Message Authentication Code: 92981e96143be2501f1bcdb6cad6c343
Windows Domain Members

5E 04 00 00

Key Selector

RID
Windows Domain Members

/* Sign the NTP response with the unicodePwd */
MD5Init(&ctx);
MD5Update(&ctx, nt_hash->hash, sizeof(nt_hash->hash));
MD5Update(&ctx, sign_request.packet_to_sign.data, 
sign_request.packet_to_sign.length);
MD5Final(signed_reply.signed_packet.data + sign_request.packet_to_sign.length + 4, &ctx);

* Username : DELOREANPC$
* Domain : PTDOM
* Password : 01 09 8b 63 35 9f 69 3d 15 9f d1 2a 03 74 ef 9b c3 70 ec 0
7 3b 5c d3 54 84 1e ca 94 94 01 b3 b7 99 0f b0 7e 88 fc 1c 10 67 f3 ee 5e f2 26
bd 1d b2 6a e1 d8 fa ff ac e7 18 32 56 35 57 6f 0b 7d a1 24 31 d7 57 88 39 84 c3
5f aa 15 df f8 6a d3 d9 35 51 15 f5 d6 26 c2 d6 c4 18 ec 0d 22 21 be 6c f2 ac 8
8 2a 95 49 92 11 b8 a6 5d 03 77 aa 08 c6 9d 75 b4 62 0a 9a dc 6c c1 e7 7d 28 75
4c 2a 5b 44 00 19 8e bf b3 81 ca 23 31 01 e5 aa 14 c2 28 8c 71 9b a0 8b 9f ad 47
be 53 7f e9 b4 e1 21 8f ff 82 11 4b cd e8 d6 d0 b7 8d b8 e2 69 08 42 e3 0a 3c 3
9 6c 61 97 3c cb e8 e5 2b bd 1b 33 c6 55 08 1c 3e d5 49 d3 b1 20 93 9f ed 27 dd
82 eb c4 26 15 30 3b d3 0a 76 df 75 52 61 c8 76 9f 22 a2 aa d0 39 49 27 35 46 22
80 9e 59 f9 d7 80 9f
## Windows Domain Members

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00000000000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>2</td>
<td>12.654537000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>3</td>
<td>22.538317000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>4</td>
<td>32.064646000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>5</td>
<td>32.132393000</td>
<td>192.168.2.2</td>
<td>192.168.1.100</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, server</td>
</tr>
<tr>
<td>6</td>
<td>41.243363000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>7</td>
<td>51.360859000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>8</td>
<td>60.192576000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>9</td>
<td>71.125885000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>10</td>
<td>80.917164000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>11</td>
<td>89.873160000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>12</td>
<td>99.663807000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>13</td>
<td>108.534417000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>14</td>
<td>119.530028000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>15</td>
<td>128.487563000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>16</td>
<td>128.525009000</td>
<td>192.168.2.2</td>
<td>192.168.1.100</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, server</td>
</tr>
<tr>
<td>17</td>
<td>151.116206000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
<tr>
<td>18</td>
<td>151.118500000</td>
<td>192.168.2.2</td>
<td>192.168.1.100</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, server</td>
</tr>
<tr>
<td>19</td>
<td>158.994790000</td>
<td>192.168.1.100</td>
<td>192.168.2.2</td>
<td>NTP</td>
<td>110</td>
<td>NTP Version 3, client</td>
</tr>
</tbody>
</table>
Not a silver bullet
Let’s Go!

• Modern Time Synchronisation
• Get in a Delorean
• HTTP Strict Transport Security
• Windows task scheduler
• Public Key Infrastructure
• Conclusions & Recommendations
Delorean

  - [http://github.com/PentesterES/Delorean](http://github.com/PentesterES/Delorean)

- Based on a kimifly’s work:
  - [http://github.com/lifly/ntpserver](http://github.com/lifly/ntpserver)

- Implements several attacks.
- It pretends to be an NTP attack ‘suite’.
Delorean

$ ./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
Basic attacks

# ./delorean.py -n

# ./delorean.py -s 10d -n

# ./delorean.py -d ‘2020-08-01’ -n

# ./delorean.py -r -n
DEMO
Time Skimming Attack

315,360 secs

later
# ./delorean.py -k 15h -t 10s -n
Replay Attack

```
$ ./delorean.py -n -r capture.pcap
```

Network Time Protocol (NTP Version 3, server)

Flags: 0x1c
Peer Clock Stratum: primary reference (1)
Peer Polling Interval: 17 (131072 sec)
Peer Clock Precision: 0.015625 sec
Root Delay: 0.00000 sec
Root Dispersion: 10.8970 sec
Reference ID: uncalibrated local clock
Transmit Timestamp: Oct 7, 2014 08:28:38.118040000 UTC
Key ID: 5e040000
Message Authentication Code: 92981e96143be2501f1bcdb6cad6c343
Spoofing Attack

$ ./delorean.py -n -f 192.168.10.10 -o 8.8.8.8 -r capture.pcap
Flooding to 192.168.10.10

$ tcpdump -nn -p -i eth1 host 192.168.10.10
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
08:26:07.621412 IP 8.8.8.8.123 > 192.168.10.10.123: NTPv4, Server, length 48
08:26:07.766434 IP 8.8.8.8.123 > 192.168.10.10.123: NTPv4, Server, length 48
08:26:07.843923 IP 8.8.8.8.123 > 192.168.10.10.123: NTPv4, Server, length 48
08:26:07.905666 IP 8.8.8.8.123 > 192.168.10.10.123: NTPv4, Server, length 48
Anti replaying...

Network Time Protocol (NTP Version 4, server)

Flags: 0x24
- Leap Indicator: no warning (0)
- Version number: NTP Version 4 (4)
- 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 08:40:04.634295000 UTC
Let’s Go!

• Modern Time Synchronisation
• Get in a Delorean
• HTTP Strict Transport Security
• Windows task scheduler
• Public Key Infrastructure
• Conclusions & Recommendations
Stripping SSL links

GET / HTTP/1.1

<body>
<img src=whatever.jpg>
<a href= http://myweb/login>
</body>
HTTP Strict Transport Security

- 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?
How it work?

Strict-Transport-Security: max-age=3153600

GET / HTTP/1.1

HTTPS

Server

Client
Parameters

- **max-age**: amount of seconds that the policy is enabled.
- **includeSubdomains**: If present, the policy applies to all subdomains, not just the visited one.

```bash
$ ./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
```
Browsers support

- http://caniuse.com/#feat=stricttransportsecurity
HSTS Timeline

HTTPS connection 3153600 secs later
Preloaded HSTS

• Hardcoded list of well known website names that should always use HTTPS.

• Prevent the security gap before the first HTTPS connection.

• Google, Twitter, Paypal, ...
Avoid protected names
HTTPS connection
3153600 secs later
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.

https://blog.mozilla.org/security/2012/11/01/preloading-hsts/
Currently the HSTS "max-age" value is four hours. We already aware of this and we have an existing plan to increase this value in near future. Additionally, Chrome and Firefox come with pre-loaded lists of popular websites (Including) for which HSTS is enforced by default.

The real world attack window is negligible to conduct Man-In-Middle (MIM) by taking advantage of HSTS low "max-age" value. Because the MIM should target a victim with the condition which includes the following condition:

- Victim should be connecting to [redacted] for the first time after 4 hours of his last [redacted] access + Victim should type [blue]http://[redacted] instead of [redacted] + Victim should be using browser other than Google Chrome and Mozilla Firefox.

In addition to that, every web page in [redacted] domain is rendered over HTTPS and we have extensive risk detection to identify and prevent malicious transaction/activities. For all these reasons, We find the risk to be negligible in both its assertion as well as our practical experience.
Chromium Source Code

```cpp
transport_security_state.cc

}  
DCHECK(result.domain_id != DOMAIN_NOT_PINNED);

UMA_HISTOGRAM_SPARSE_SLOWLY(
  "Net.PublicKeyPinFailureDomain", result.domain_id);

// static
bool TransportSecurityState::IsBuildTimely() {
  // If the build metadata aren't embedded in the binary then we can't use the
  // build time to determine if the build is timely, return true by default. If
  // we're building an official build then keep using the build time, even if
  // it's invalid it'd be a date in the past and this function will return
  // false.
  #if defined(DONT_EMBED_BUILD_METADATA) && !defined(OFFICIAL_BUILD)
    return true;
  #else
    const base::Time build_time = base::GetBuildTime();
    // We consider built-in information to be timely for 10 weeks.
    return (base::Time::Now() - build_time).InDays() < 70 /* 10 weeks */;
  #endif
}
```
$ 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" => [...]"
HSTS weakness

• Its security relies on time.

• It completely trust the OS’s current time.

• This looks like a job for Delorean!
DEMO
Public release
From: Adam <adam@google.com>
Date: Thu, 16 Oct 2014 14:28:38 -0700
Message-ID: <CAL9PXLx4yhzq37Mwfnh4Vss6xZbg1e5ogq9eq_9WQXOtig@mail.gmail.com>
To: Anne <anne@example.com>, "public-webappsec@w3.org" <public-webappsec@w3.org>
Cc: John <john@example.com>

Someone pointed out that the author did a demo so there must be something there.

So I went back into the source code and the author really is mistaken by the 1000 days bit in net-internals. However, we do have a timeout for HSTS preloads which git blame says that I added, although I don’t remember it. The timeout is the same as our pinning timeout, which is 10 weeks from the build timestamp.

This is the motivation for things like tlsdate (https://github.com/ioerror/tlsdate) as used in parts of ChromeOS.

However, in section seven, where the author claims that preloaded entries are added for 1000 days, that’s only via the net-internals debugging interface. (The code screenshot shown is also of code for that debugging interface.) I believe that preloaded entries in Chrome will always be enforced, no matter what the system time is.

Cheers
Subject: Re: HSTS Attack Demo
From: Adam <adam@google.com>
Date: 17/10/2014 19:34
To: Adrienne <adrienne@google.com>
CC: Chris <chris@google.com>, Jose Selvi <j.selvi@pentester.es>

On Thu, Oct 16, 2014 at 9:29 PM, Adrienne <adrienne@google.com> wrote:
| Is there a reason why pre-loaded HSTS rules expire?

So that we can effectively actually remove entries when needed.

I agree that it's not a big deal and I'd also be ok with them not expiring. But I think that would be papering over a crack -- lots of things goes wrong when the clock is off.

Cheers
Let’s Go!

• Modern Time Synchronisation
• Get in a Delorean
• HTTP Strict Transport Security
• Windows task scheduler
• Public Key Infrastructure
• Conclusions & Recommendations
Task scheduler

- **SynchronizeTime**
  - **Description:** Maintains date and time synchronization on all clients and servers in the network. If this service is stopped, date and time synchronization will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start.
Windows automatic updates

![Windows Update configuration screen]

- **Name**: Scheduled Start
- **Location**: \Microsoft\Windows\WindowsUpdate
- **Author**: Microsoft Corporation
- **Description**: This task is used to start the Windows Update service when needed to perform scheduled operations such as scans.

**Scheduled Start properties**
- **Triggers**: Multiple triggers defined
- **Next Run Time**: 2/2/2016 4:09:50 AM
- **Last Run Time**: Never

**Security options**:
- **When running the task, use the following user account**: SYSTEM
  - **Run only when user is logged on**
  - **Do not store password. The task will only have access to local resources**
  - **Run with highest privileges**
Let’s Go!

• Modern Time Synchronisation
• Get in a Delorean
• HTTP Strict Transport Security
• Windows task scheduler
• Public Key Infrastructure
• Conclusions & Recommendations
PKI, CAs & Certificates
Edo Tensei no Jutsu!
Weak certificates

https://www.eff.org/observatory
Looking around Las Vegas
Let’s look any other...
DEMO
Certificate:
Data:
   Version: 3 (0x2)
   Serial Number:
   Signature Algorithm: sha1WithRSAEncryption
   Issuer:
      emailAddress     = info@diginotar.nl
      commonName       = DigiNotar Public CA 2025
      organizationName = DigiNotar
      countryName      = NL
   Validity
      Not Before: Jul 10 19:06:30 2011 GMT
      Not After : Jul  9 19:06:30 2013 GMT
   Subject:
      commonName       = *.google.com
      serialNumber     = PK000229200002
      localityName     = Mountain View
      organizationName = Google Inc
      countryName      = US
   Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (2048 bit)
         Modulus (2048 bit):
Heartbleed

SERVER, ARE YOU STILL THERE? IF SO, REPLY “HAT” (500 LETTERS).

User Meg wants these 500 letters: HAT.

HAT. Lucas requests the “missed connections” page. Eve (administrator) wants to set server’s master key to “148 35038534”. Isabel wants pages about “snakes but not too long”. Jane Karen wants to change account password to “Patched”. User labor requests more.
# Debian PRNG

```java
int getRandomeNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Certificate Chain

.EXPIRED

.EXPIRED
Revocation lists
An example...
X.509 defines one method of certificate revocation. This method involves each CA periodically issuing a signed data structure called a certificate revocation list (CRL). A CRL is a time stamped list identifying revoked certificates which is signed by a CA or CRL issuer and made freely available in a public repository. Each revoked certificate is identified in a CRL by its certificate serial number. When a certificate-using system uses a certificate (e.g., for verifying a remote user's digital signature), that system not only checks the certificate signature and validity but also acquires a suitably-recent CRL and checks that the certificate serial number is not on that CRL. The meaning of "suitably-recent" may vary with local policy, but it usually means the most recently-issued CRL. A new CRL is issued on a regular periodic basis (e.g., hourly, daily, or weekly). An entry is added to the CRL as part of the next update following notification of revocation. An entry MUST NOT be removed from the CRL until it appears on one regularly scheduled CRL issued beyond the revoked certificate's validity period.
## Purged CRLs???

<table>
<thead>
<tr>
<th>CRL</th>
<th>ID más viejo</th>
<th>Fecha</th>
</tr>
</thead>
<tbody>
<tr>
<td>DigiCert SHA2 Extended Validation Server CA</td>
<td>131213031902Z0 (330 certs)</td>
<td>13/12/2013 03:19</td>
</tr>
<tr>
<td>(Dropbox, GitHub)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DigiCert High Assured CA 3</td>
<td>120614172516Z0 140927190602Z0</td>
<td>14/06/2012 17:25</td>
</tr>
<tr>
<td>(Facebook)</td>
<td></td>
<td>27/09/2014 19:06</td>
</tr>
<tr>
<td>GeoTrust Global CA (Google)</td>
<td>020521134804Z0 (9 certs)</td>
<td>21/05/2002 13:48</td>
</tr>
<tr>
<td>GlobalSign Organization Validation CA -</td>
<td>140331025038Z0 (637 certs)</td>
<td>31/03/2014 02:50</td>
</tr>
<tr>
<td>SHA256 - G2 (LogmeIn)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VeriSign Class 3 Extended Validation SSL CA</td>
<td>121204020253Z0 (1709 certs)</td>
<td>04/12/2012 02:02</td>
</tr>
<tr>
<td>(Microsoft, Paypal, Twitter)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VeriSign Class 3 Secure Server CA - G3 (Yahoo)</td>
<td>101010055242Z0 (41120 certs)</td>
<td>10/10/2010 05:52</td>
</tr>
</tbody>
</table>
Online Certificate Status Protocol

1. The CA periodically publishes its CRLs to the OCSP responder.
2. A client contacts the OCSP to see if a cert is valid.
3. The OCSP replies with the cert status.

User

OCSP Responder

CRL

Certificate Manager

nccgroup
What if I can’t connect?

https://www.grc.com/revocation/implementations.htm
DEMO
Let’s Go!

- Modern Time Synchronisation
- Get in a Delorean
- HTTP Strict Transport Security
- Windows task scheduler
- Public Key Infrastructure
- Conclusions & Recommendations
Conclusions & Recommendations

Facts

- Time synchronisation isn’t managed securely by most operating system vendors.
- Many security protections relies in time. If an attacker can control the local clock, lots of things can go wrong.

What to do

- Configure NTP synchronisation in a secure way (Microsoft does):
  - Signature.
  - Maximum drift.
- Block SSL certificates which expiry date is before the browser build date or the last update (Chrome does).
Thanks! Questions?

Jose Selvi
http://twitter.com/JoseSelvi

Jose.Selvi@nccgroup.trust
http://www.nccgroup.trust

jselvi@pentester.es
http://www.pentester.es
UK Offices
Manchester - Head Office
Cheltenham
Edinburgh
Leatherhead
London
Thame

North American Offices
San Francisco
Atlanta
New York
Seattle

Australian Offices
Sydney

European Offices
Amsterdam - Netherlands
Munich – Germany
Zurich - Switzerland