HTTPS in the real world

Joe DeBlasio
jdeblasio@chromium.org
@deblasioj

Who am I?

I lead Chrome Security's "HTTPS Ecosystem Engineering" team.

We work on:

- increasing HTTPS adoption,
- communicating site identity to users,
- researching on how people use Chrome’s security features,
- behind-the-scenes ecosystem stuff,
  (e.g. Certificate Transparency, HSTS, TLS deprecations, etc.), and

I also do other security team work:

- security reviews and consulting for other Chrome teams.
- Chrome's Vulnerability Rewards Program

Previously: PhD in e-crime and web security measurement from UCSD.

Reminder: **HTTPS**

https:// provides **confidentiality**, **integrity**, **authentication**.

But

1. not all sites **support** https://
2. *even when* sites support https://, we still sometimes still use http://
3. https:// is only as strong as the certificate

Today: attempts to fix (2) and (3),
with lessons about what worked in the real world and what didn't.

HTTPS sites don't always use HTTPS

Always HTTPS: **users still end up on http://**

Even when sites fully support* https://, users still use http:// sometimes:

- User clicks on http:// links
- User types “example.com” into the address bar**
- https:// page loads http:// subresources** (e.g. images)

* That's not guaranteed. Again, a different talk.

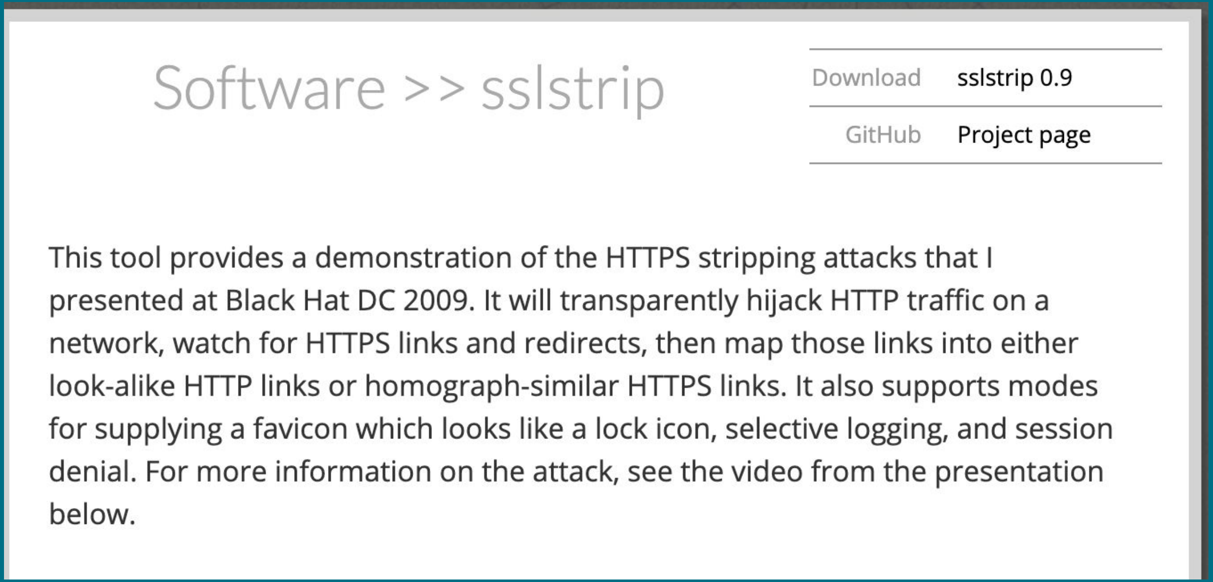
** We've partially fixed these in the last ~year.

Always HTTPS: **Good sites redirect users**

```
$ curl -v http://joedeblasio.com/foo
> GET /foo HTTP/1.1
> Host: joedeblasio.com
>
< HTTP/1.1 301 Moved Permanently
< Location: https://joedeblasio.com/foo
```

Not good enough! Attackers can prevent these redirects!

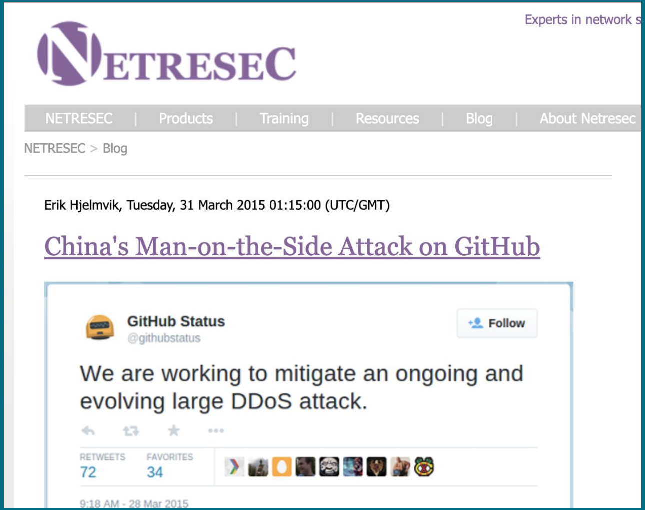
Always HTTPS: **http:// interception happens**



Software >> sslstrip

Download	sslstrip 0.9
GitHub	Project page

This tool provides a demonstration of the HTTPS stripping attacks that I presented at Black Hat DC 2009. It will transparently hijack HTTP traffic on a network, watch for HTTPS links and redirects, then map those links into either look-alike HTTP links or homograph-similar HTTPS links. It also supports modes for supplying a favicon which looks like a lock icon, selective logging, and session denial. For more information on the attack, see the video from the presentation below.



Experts in network s

NETRESEC

NETRESEC | Products | Training | Resources | Blog | About Netresec

NETRESEC > Blog

Erik Hjelmvik, Tuesday, 31 March 2015 01:15:00 (UTC/GMT)

China's Man-on-the-Side Attack on GitHub

GitHub Status
@githubstatus

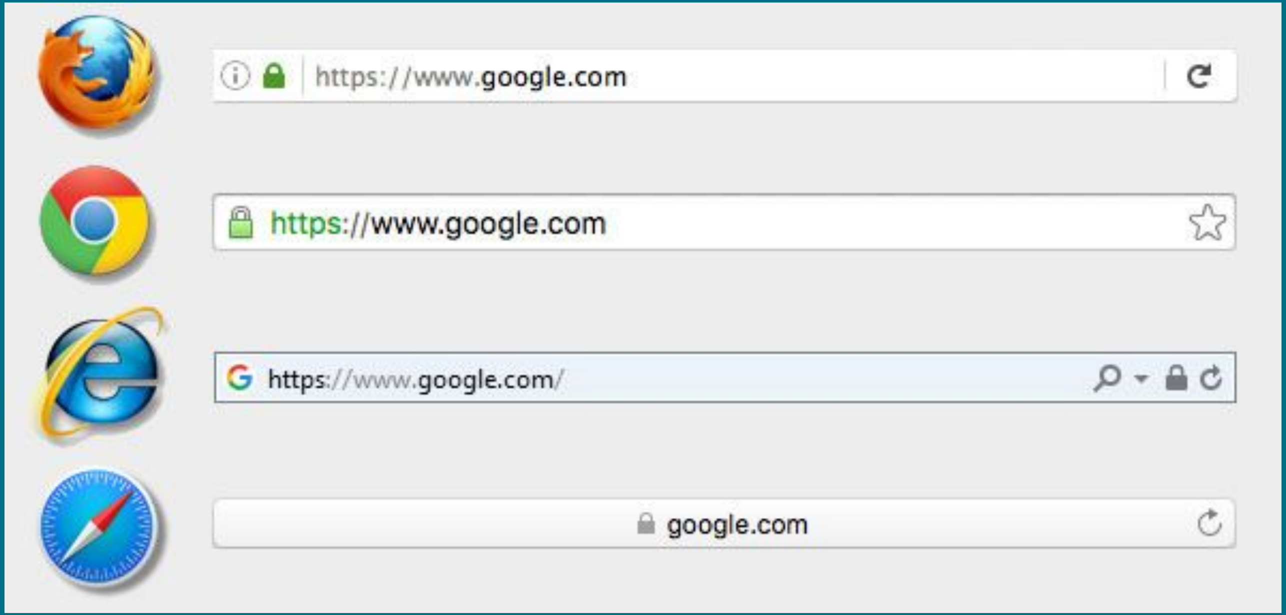
Follow

We are working to mitigate an ongoing and evolving large DDoS attack.

RETWEETS 72 FAVORITES 34

9:18 AM - 28 Mar 2015

Always HTTPS: **fix attempt - positive UI**



Always HTTPS: **fix attempt - positive UI**

Doesn't work!

- People don’t notice missing indicators.
- Many don't know what they mean.
- Not actionable. What's the user supposed to do?

See

- “The Emperor’s New Security Indicators” (Schechter et al.)
- “An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks” (Jackson et al.)
- “‘If HTTPS Were Secure, I Wouldn’t Need 2FA’ -- End User and Administrator Mental Models of HTTPS” (Krombholz et al.)
- “The Web's Identity Crisis: Understanding the Effectiveness of Website Identity Indicators” (Thompson et al.)

Always HTTPS: **Strict Transport Security**

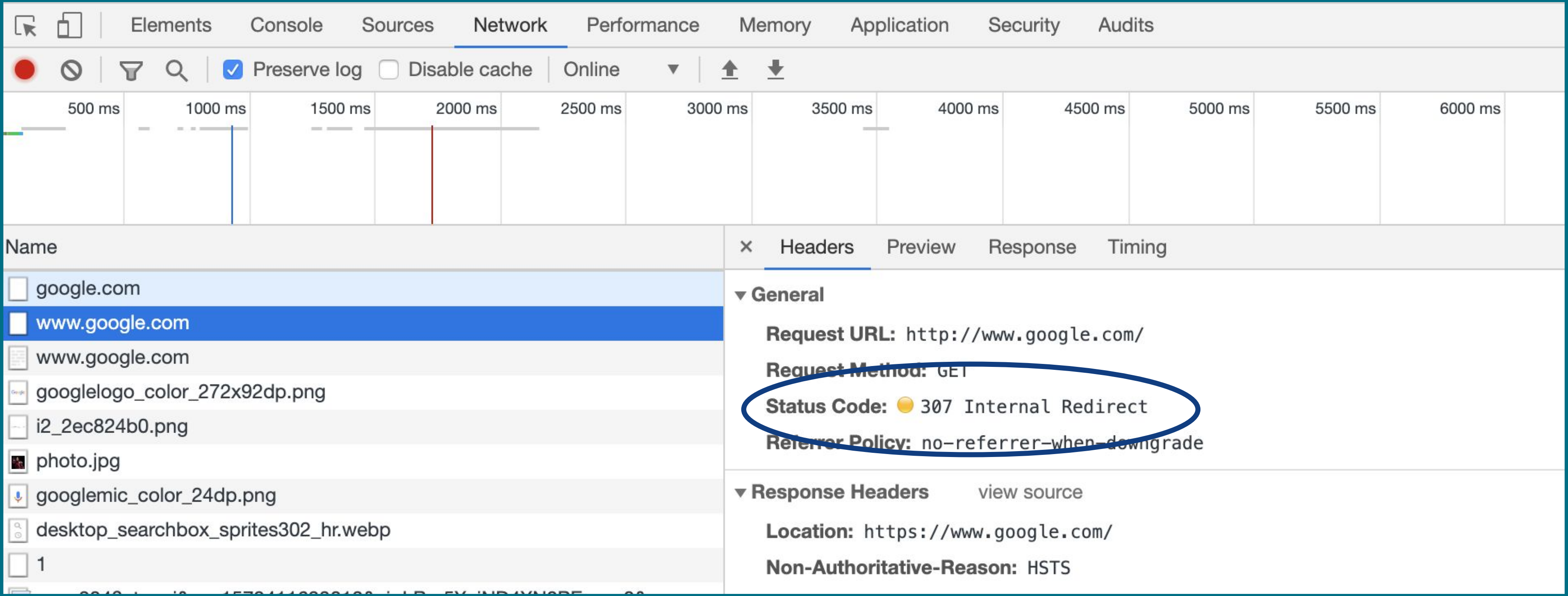
Let websites opt-in to strict mode: “Only ever contact me via https://.”

Always HTTPS: **Strict Transport Security**

Let websites opt-in to strict mode: “Only ever contact me via https://.”

- Browser transparently **rewrites http**://example.com to **https**://example.com.
- Invalid certificate on https://example.com **can't be bypassed**.

HSTS = HTTP Strict Transport Security



Name

- google.com
- www.google.com
- www.google.com
- googlelogo_color_272x92dp.png
- i2_2ec824b0.png
- photo.jpg
- googlemic_color_24dp.png
- desktop_searchbox_sprites302_hr.webp
- 1

General

Request URL: `http://www.google.com/`

Request Method: `GET`

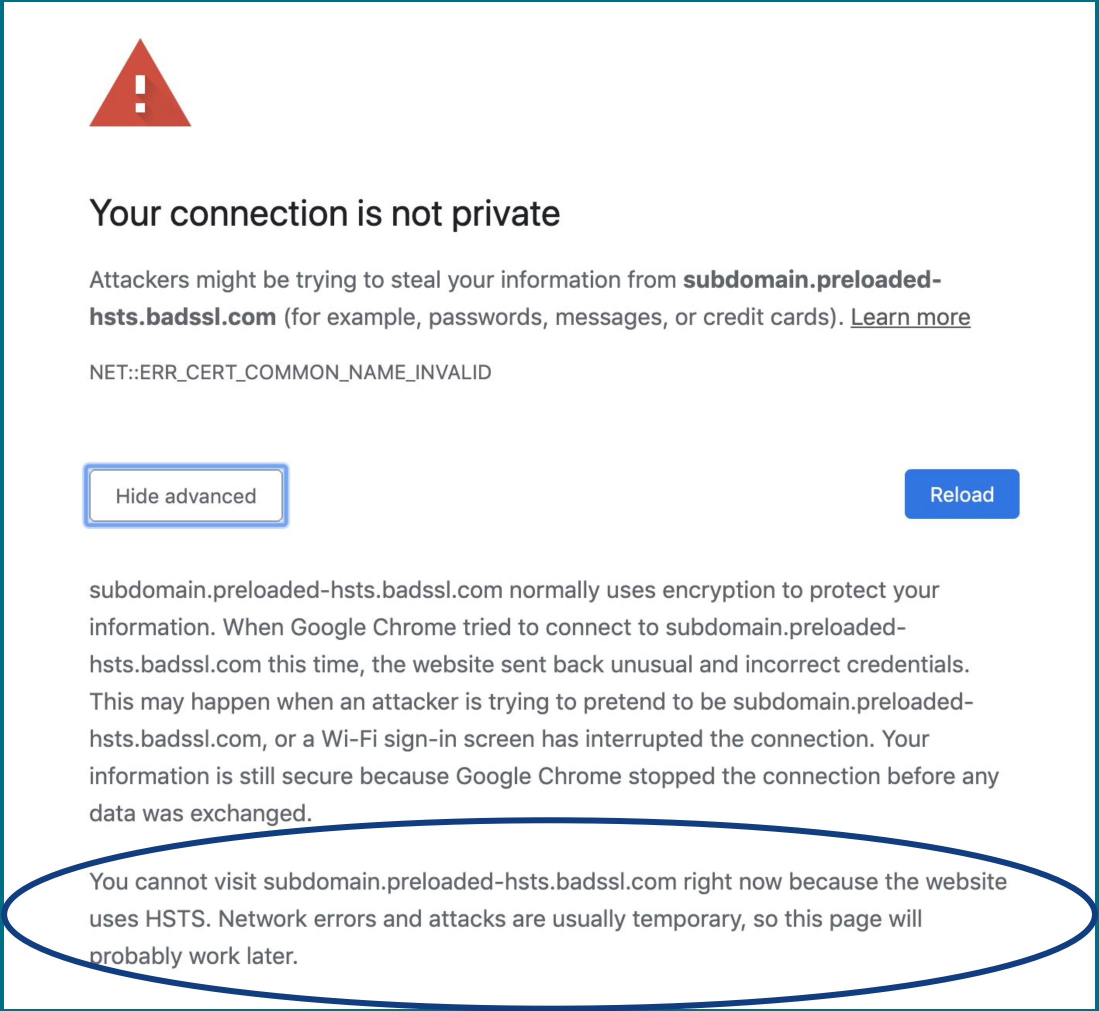
Status Code: ● `307 Internal Redirect`

Referrer Policy: `no-referrer-when-downgrade`

Response Headers view source

Location: `https://www.google.com/`

Non-Authoritative-Reason: `HSTS`



Attackers might be trying to steal your information from **subdomain.preloaded-hsts.badssl.com** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_COMMON_NAME_INVALID

subdomain.preloaded-hsts.badssl.com normally uses encryption to protect your information. When Google Chrome tried to connect to subdomain.preloaded-hsts.badssl.com this time, the website sent back unusual and incorrect credentials. This may happen when an attacker is trying to pretend to be subdomain.preloaded-hsts.badssl.com, or a Wi-Fi sign-in screen has interrupted the connection. Your information is still secure because Google Chrome stopped the connection before any data was exchanged.

You cannot visit subdomain.preloaded-hsts.badssl.com right now because the website uses HSTS. Network errors and attacks are usually temporary, so this page will probably work later.

Always HTTPS: **Strict Transport Security**

Opt-in to HSTS via HTTP response header:

```
    Strict-Transport-Security: max-age=<expire-time>

    Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
```

Always HTTPS: **Strict Transport Security**

Opt-in to HSTS via HTTP response header:

```
    Strict-Transport-Security: max-age=<expire-time>

    Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
```

How long (in seconds) the browser should remember this STS information

Always HTTPS: **Strict Transport Security**

Opt-in to HSTS via HTTP response header:

```
Strict-Transport-Security: max-age=<expire-time>
Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
```

Apply to all subdomains, e.g. if header observed on example.com, upgrade foo.example.com, too.

Always HTTPS: **Strict Transport Security**

HSTS Gotchas:

Always HTTPS: **Strict Transport Security**

HSTS Gotchas:

- No way to set `includeSubdomains` for all-but-a-few subdomains
    - Hard for big organizations with many subdomains operated separately

Always HTTPS: **Strict Transport Security**

HSTS Gotchas:

- No way to set `includeSubdomains` for all-but-a-few subdomains
    - Hard for big organizations with many subdomains operated separately
- No official way to set HSTS on *full domain* from *subdomain*
    - e.g., users visit www.example.com; site wants HSTS for all of example.com

Always HTTPS: **Strict Transport Security**

HSTS Gotchas:

- No way to set `includeSubdomains` for all-but-a-few subdomains
  - Hard for big organizations with many subdomains operated separately
- No official way to set HSTS on *full domain* from *subdomain*
  - e.g., users visit www.example.com; site wants HSTS for all of example.com
- No way to *undo* HSTS besides waiting
  - "Oh no! We forgot about that service!"

Always HTTPS: **Strict Transport Security**

HSTS Gotchas:

- No way to set `includeSubdomains` for all-but-a-few subdomains
  - Hard for big organizations with many subdomains operated separately
- No official way to set HSTS on *full domain* from *subdomain*
  - e.g., users visit www.example.com; site wants HSTS for all of example.com
- No way to *undo* HSTS besides waiting
  - "Oh no! We forgot about that service!"
- Doesn’t protect first visit
  - HSTS is delivered via header sent with HTTPS connection. Chicken and egg problem.

Always HTTPS: **Strict Transport Security**

Biggest HSTS Gotcha: Using HSTS for tracking

- Sites can set and read **persistent state** from a 3rd-party context
- That's a "supercookie"!
    - Can be used to track users
    - Can’t be viewed, restricted, or cleared by users

Always HTTPS: **Strict Transport Security**

Setting the supercookie:

1. Users visits shopping-site.com
2. shopping-site.com loads script from ad-network.com
3. ad-network.com script assigns user a unique ID (say, 0b11010001), and loads subresources for each bit set in the identifier:
   - 1.ad-network.com
   - 5.ad-network.com
   - 7.ad-network.com
   - 8.ad-network.com
4. Each subresource sets HSTS for that subdomain

Always HTTPS: **Strict Transport Security**

Reading the supercookie:

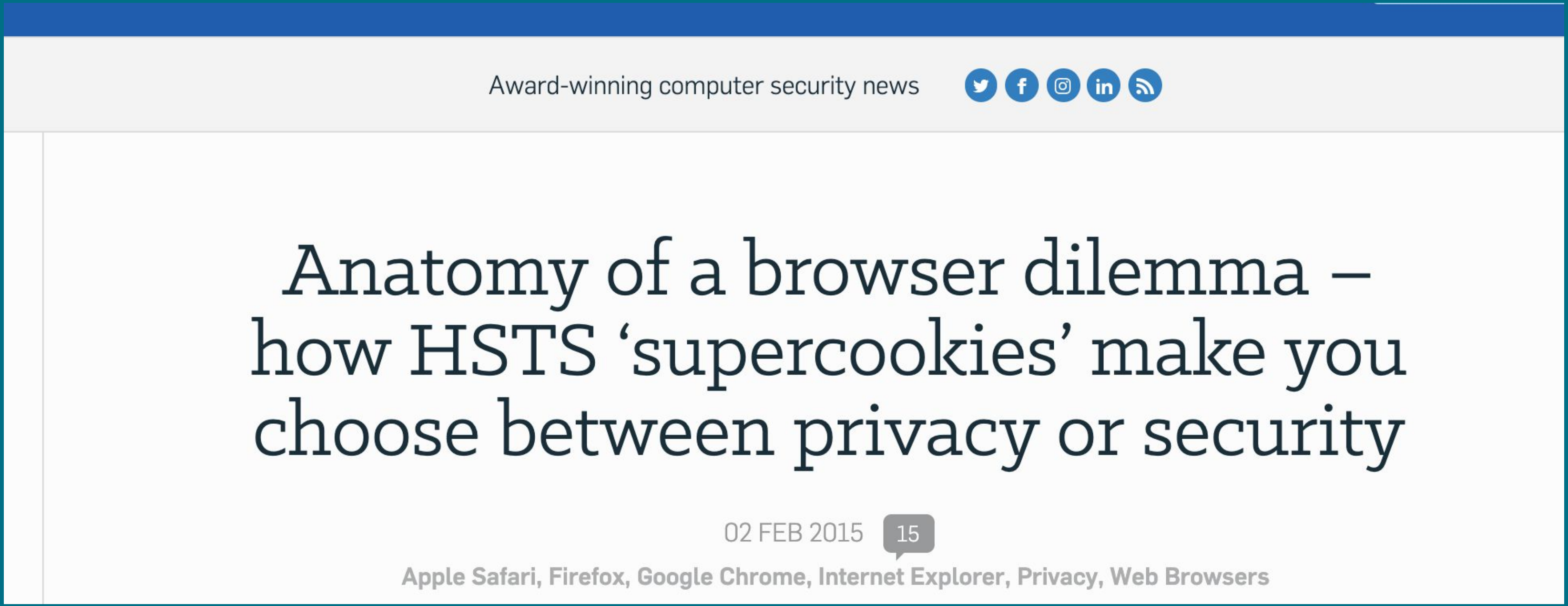
- Users visits news-site.com, loads analytics script from ad-network.com
- ad-network.com script loads subresource for each bit
  - 1.ad-network.com
  - 2.ad-network.com
  - ...
  - 8.ad-network.com
- ad-network.com observes which subresources redirect to https://, reconstructs ID

Always HTTPS: **Strict Transport Security**

“This information is cached in the HSTS Policy store… This information can be retrieved by other hosts through cleverly constructed and loaded web resources… Such a technique could potentially be abused as yet another form of ‘web tracking.’”

- https://tools.ietf.org/html/rfc6797

Always HTTPS: **Strict Transport Security**



Award-winning computer security news

Anatomy of a browser dilemma – how HSTS ‘supercookies’ make you choose between privacy or security

02 FEB 2015 15

Apple Safari, Firefox, Google Chrome, Internet Explorer, Privacy, Web Browsers

https://nakedsecurity.sophos.com/2015/02/02/anatomy-of-a-browser-dilemma-how-hsts-supercookies-make-you-choose-between-privacy-or-security/

Always HTTPS: **Strict Transport Security**

"Recently we became aware that this theoretical attack was beginning to be deployed against Safari users."

- "Protecting Against HSTS Abuse" (WebKit blog, March 2018)

Always HTTPS: **Mitigating HSTS tracking**

- No perfect solution: everything requires trade-offs of security and privacy

Always HTTPS: **Mitigating HSTS tracking**

Safari's mitigations:

- **Setting the cookie:** Allow subresources to set HSTS only for the first-party hostname or the registrable domain
  - When on **foo.bar**.example.com, subresources can set their HSTS only if they are **foo.bar**.example.com or example.com, not **bar**.example.com or **baz.foo.bar**.example.com.
  - Pop quiz: why allow registrable domain?

- **Reading the cookie:** piggyback on third-party cookie blocking
  - If Safari is blocking 3rd party cookies, ignore HSTS on subresources
  - Relies on existing complex 3rd party cookie blocking logic

Downside: you must visit a domain directly in order to set HSTS!

Always HTTPS: **Mitigating HSTS tracking**

Chrome's mitigations:

- Forbid all mixed content (http:// subresources on https:// pages)
    - A good idea regardless of HSTS tracking
- Do not apply HSTS upgrades to subresources on http:// pages*
    - Minimal security loss from disregarding HSTS for subresources on http:// pages

Downside: doesn't protect HTTPS subresources at all (though that's kinda already true)

* somewhat tentative

Always HTTPS: **Mitigating HSTS tracking**

Firefox's mitigations:

- Partition HSTS state by domain name at the top-level
  - e.g. when on foo.example.com, don't apply HSTS state for subresources that you learned when on bar.example.com

Downside: limits when you "remember" HSTS, exacerbating first-visit problem

Always HTTPS: **Strict Transport Security**

HSTS Gotchas:

- No way to set `includeSubdomains` for all-but-a-few subdomains
  - Hard for big organizations with many subdomains operated separately
- No official way to set HSTS on *full domain* from *subdomain*
  - e.g., users visit www.example.com; site wants HSTS for all of example.com
- No way to *undo* HSTS besides waiting
  - "Oh no! We forgot about that service!"
- Doesn’t protect first visit
  - HSTS is delivered via header sent with HTTPS connection. Chicken and egg problem.

Always HTTPS: **Strict Transport Security**

Opt-in to HSTS via HTTP response header:

```
Strict-Transport-Security: max-age=<expire-time>

Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
```

Always HTTPS: **Strict Transport Security**

Opt-in to HSTS via HTTP response header:

```
    Strict-Transport-Security: max-age=<expire-time>

    Strict-Transport-Security: max-age=<expire-time>; includeSubDomains

    Strict-Transport-Security: max-age=<expire-time>; preload
```

Always HTTPS: **Strict Transport Security**

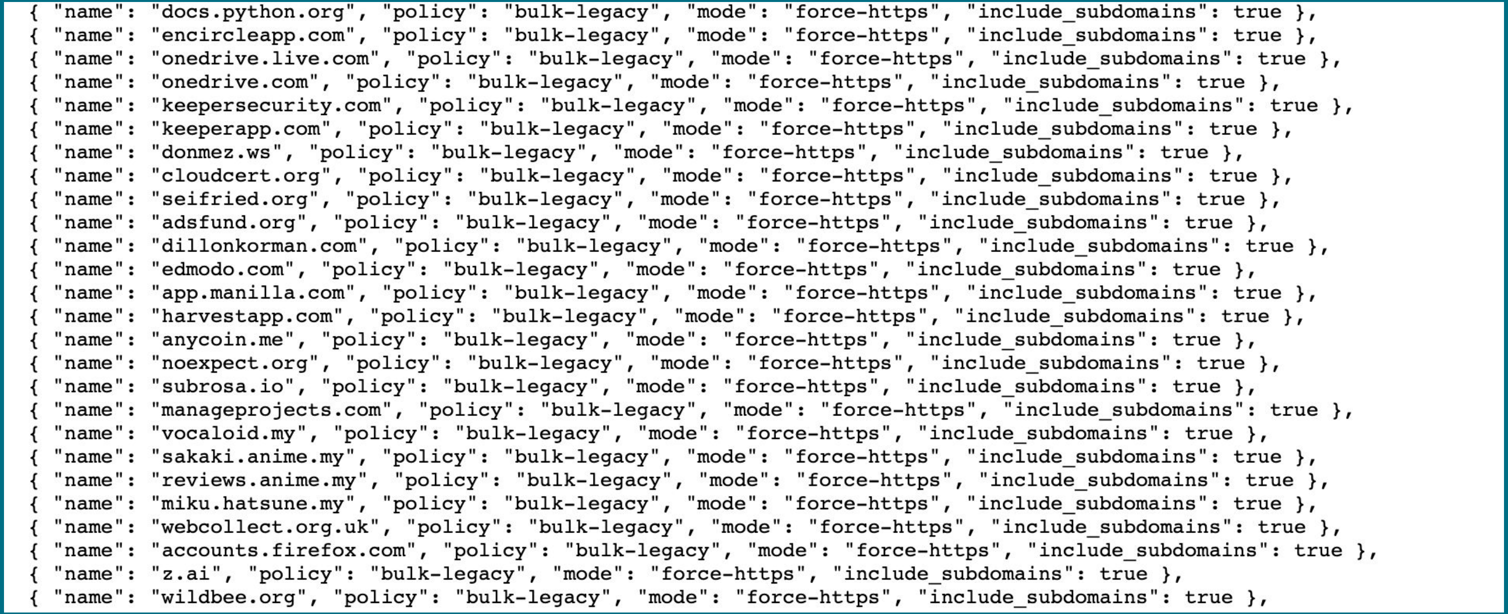
Opt-in to HSTS via HTTP response header:

```
Strict-Transport-Security: max-age=<expire-time>
Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
Strict-Transport-Security: max-age=<expire-time>; preload
```

Allow browsers to include your HSTS state before a user visits (e.g. in their source code).

Always HTTPS: **HSTS Preload**

Browsers ship **baked-in** lists of HSTS sites



```
{ "name": "docs.python.org", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "encircleapp.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "onedrive.live.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "onedrive.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "keepersecurity.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "keeperapp.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "donmez.ws", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "cloudcert.org", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "seifried.org", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "adsfund.org", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "dillonkorman.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "edmodo.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "app.manilla.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "harvestapp.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "anycoin.me", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "noexpect.org", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "subrosa.io", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "manageprojects.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "vocaloid.my", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "sakaki.anime.my", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "reviews.anime.my", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "miku.hatsune.my", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "webcollect.org.uk", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "accounts.firefox.com", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "z.ai", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
{ "name": "wildbee.org", "policy": "bulk-legacy", "mode": "force-https", "include_subdomains": true },
```

Always HTTPS: **HSTS Preload**

List maintained by Chromium, pulled into other browsers with extra policies

- Owners submit sites at hstspreload.org
- Must serve HSTS header with `preload`, `includeSubdomains`, `max-age` >= 1 year
- Can also check for removal

Operational nightmare

- Getting *off* the list means waiting ~6 months (until all browsers have updated)
- List size grows forever
- Frequent one-off requests are handled manually

Always HTTPS: **HSTS Preload**

How do we get rid of the preload list?

- Move *all* websites to https://; deprecate http://?
- Assume all websites are https://; show warnings before using http://?
- Define “high value” sites and limit list to those sites?
- Fetch portions of list on demand?
- ...

HTTPS

1. not all sites **support** https://
2. *even when* sites support https://, we still sometimes still use http://
3. https:// is only as strong as the certificate

Stopping Malicious Certificates

Or,

“How we spent years building a thing, only to realize it was terrible and delete it later."

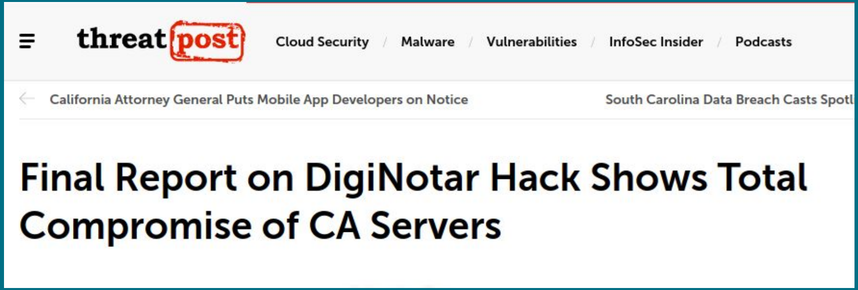
Problem: **any CA can issue cert for any site**

This is good:
website operators have supplier diversity.

Problem: **any CA can issue cert for any site**

This is good:
website operators have supplier diversity.

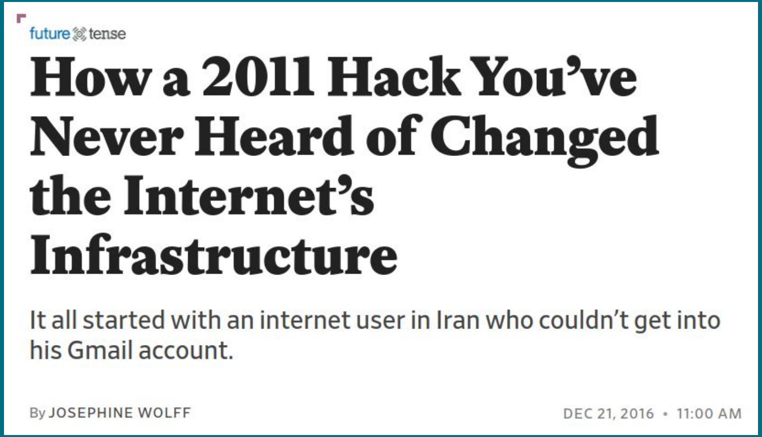
This is bad:
attackers have supplier diversity, too.



threatpost | Cloud Security / Malware / Vulnerabilities / InfoSec Insider / Podcasts

California Attorney General Puts Mobile App Developers on Notice   South Carolina Data Breach Casts Spotl

Final Report on DigiNotar Hack Shows Total Compromise of CA Servers

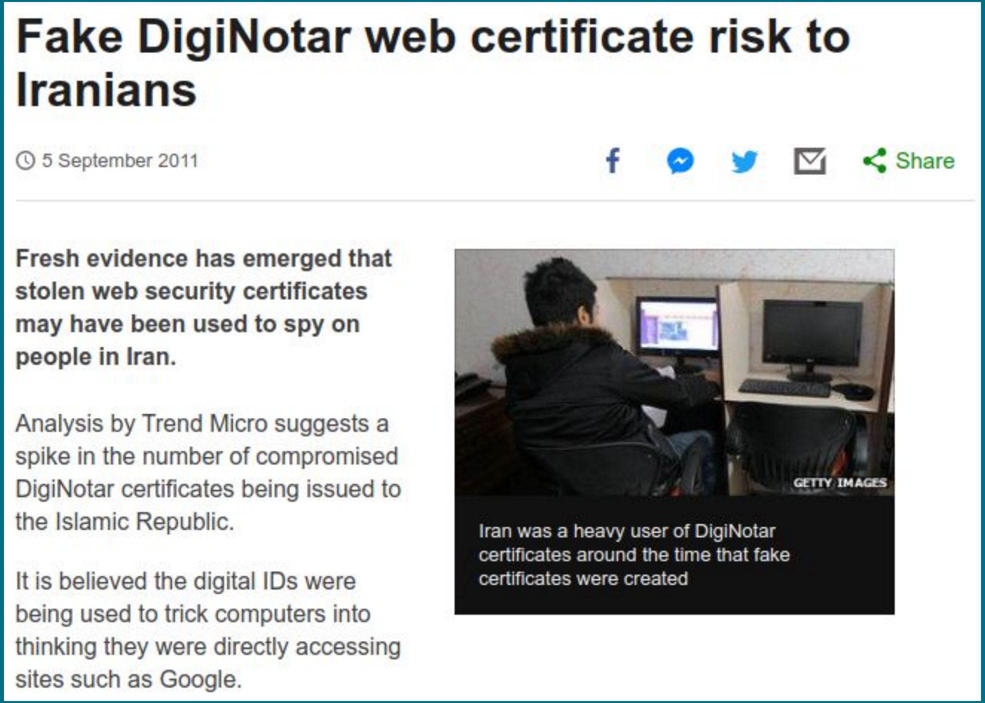


future tense

How a 2011 Hack You've Never Heard of Changed the Internet's Infrastructure

It all started with an internet user in Iran who couldn't get into his Gmail account.

By JOSEPHINE WOLFF   DEC 21, 2016 • 11:00 AM



Fake DigiNotar web certificate risk to Iranians

5 September 2011   Share

Fresh evidence has emerged that stolen web security certificates may have been used to spy on people in Iran.

Analysis by Trend Micro suggests a spike in the number of compromised DigiNotar certificates being issued to the Islamic Republic.

It is believed the digital IDs were being used to trick computers into thinking they were directly accessing sites such as Google.

Iran was a heavy user of DigiNotar certificates around the time that fake certificates were created

Stopping Malicious Certs: **borders and boundaries?**

Possible solution: Only let CAs from country X issue to websites based in country X

- But, nothing specific to DigiNotar/the Netherlands/Iran/Google about this hack*.
- The web is world-wide! We want everyone to be able to talk to everyone!
- Also, how would you enforce it?

Bottom line: this doesn't help

* For more CA failures, see sslmate.com/certspotter/failures

Stopping Malicious Certs: **HPKP**

Recall: HTTPS lets CAs attest that a given **key** belongs to a given site.

Stopping Malicious Certs: **HPKP**

Recall: HTTPS lets CAs attest that a given **key** belongs to a given site.

Possible solution: Do what SSH does! Browser remembers keys, **blocks** if key changes!

- Doesn't require big changes to the web.
- Website operators can still use any CA.
- Attackers now need a specific key.

Enter HPKP = "HTTP Public Key Pinning"

Stopping Malicious Certs: **HPKP**

The server sends an HTTP response header describing its **pin set**:

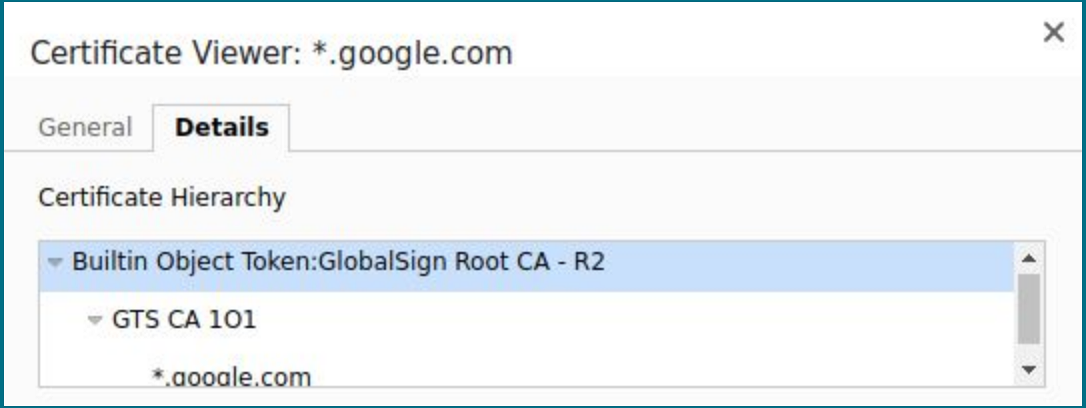
```
Public-Key-Pins: max-age=3000;
pin-sha256="d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM=";
pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g="
```

(These are `SHA256(certificate.subjectPublicKeyInfo)`, which includes the pub. key and the key type.)

Stopping Malicious Certs: **HPKP**

Site operator can pin to keys anywhere in the chain (CA to Leaf).

HPKP passes if any pin matches any key in chain.


Certificate Viewer: *.google.com

General **Details**

Certificate Hierarchy

- Builtin Object Token:GlobalSign Root CA - R2
  - GTS CA 101
    - *.google.com

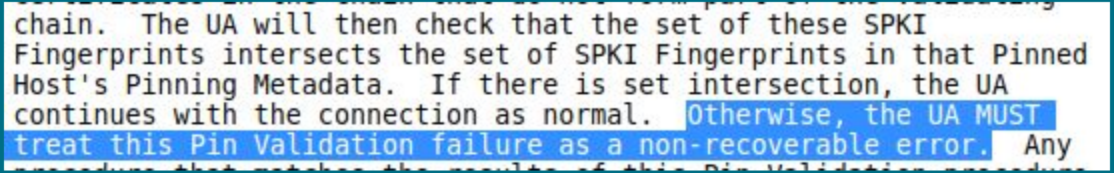
Stopping Malicious Certs: **HPKP**

Problem: *really* hard for site operators to get right

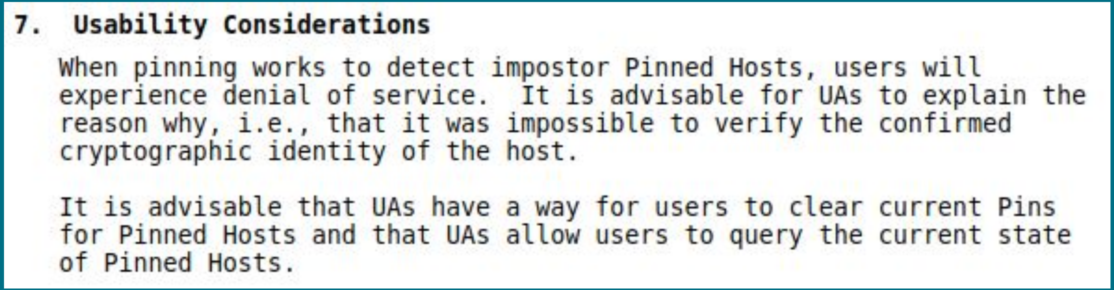
1. Almost no one understands cert chains (DAGs), issuer ecosystem, and client behavior.
2. Chain **served** ≠ chain **validated**.
3. Operators can't reliably know what chain the client will validate! It can even change!

Stopping Malicious Certs: **HPKP**

giant footgun – failure is **non-recoverable**.



chain. The UA will then check that the set of these SPKI Fingerprints intersects the set of SPKI Fingerprints in that Pinned Host's Pinning Metadata. If there is set intersection, the UA continues with the connection as normal. Otherwise, the UA MUST treat this Pin Validation failure as a non-recoverable error. Any



7. Usability Considerations

When pinning works to detect impostor Pinned Hosts, users will experience denial of service. It is advisable for UAs to explain the reason why, i.e., that it was impossible to verify the confirmed cryptographic identity of the host.

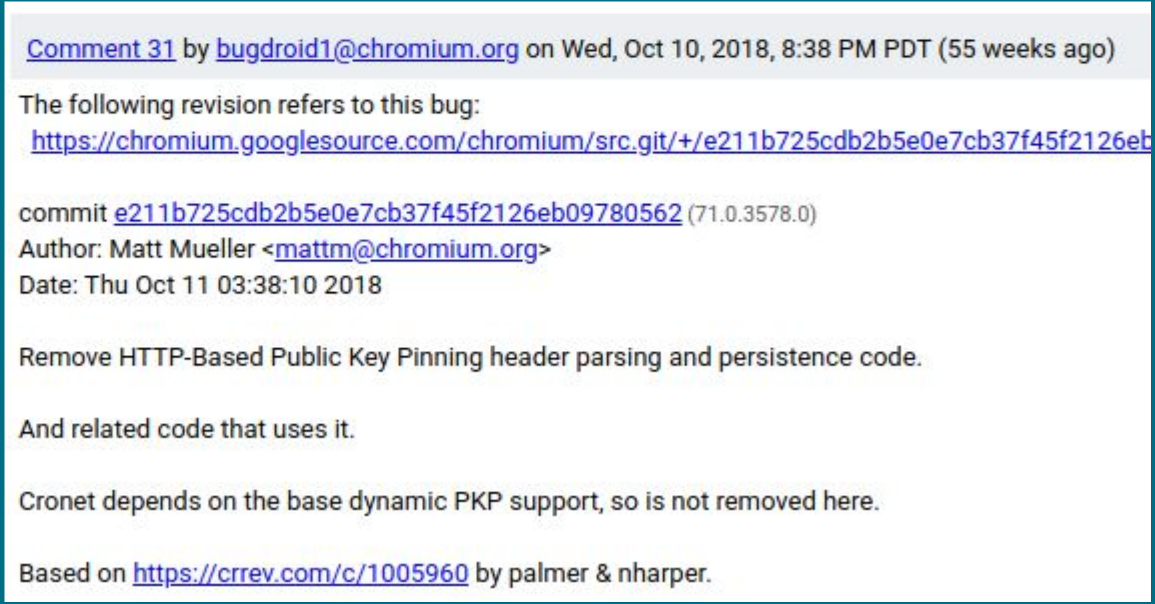
It is advisable that UAs have a way for users to clear current Pins for Pinned Hosts and that UAs allow users to query the current state of Pinned Hosts.

Related: "Hostile" pinning – attackers can DoS your service ~forever

Stopping Malicious Certs: **HPKP**

Solution: un-ship HPKP

🔥🔥🔥



Comment 31 by bugdroid1@chromium.org on Wed, Oct 10, 2018, 8:38 PM PDT (55 weeks ago)

The following revision refers to this bug:
https://chromium.googlesource.com/chromium/src.git/+/e211b725cdb2b5e0e7cb37f45f2126eb

commit e211b725cdb2b5e0e7cb37f45f2126eb09780562 (71.0.3578.0)
Author: Matt Mueller <mattm@chromium.org>
Date: Thu Oct 11 03:38:10 2018

Remove HTTP-Based Public Key Pinning header parsing and persistence code.

And related code that uses it.

Cronet depends on the base dynamic PKP support, so is not removed here.

Based on https://crrev.com/c/1005960 by palmer & nharper.

Stopping Malicious Certs: **static pinning**

Instead of HTTP response headers discovered dynamically, why not bake pins into the browser?

Stopping Malicious Certs: **static pinning**

Instead of HTTP response headers discovered dynamically,
why not bake pins into the browser?

Because it’s a major pain in the ass, that’s why. (But we still do it.)

Strength: We can manually vet "operationally-mature" orgs for inclusion.

Weakness: It doesn't scale.

Stopping Malicious Certs: **CAA**

List what CAs allowed to issue certs for a domain in a DNS record

But:

- Only advisory — enforced at ‘layer 8’ (i.e. by the CAs)
- Hard to know impact if CA ignores it.
- Relies on DNS, which isn't yet secure

```
$ dig -t caa google.com

; <<>> DiG 9.10.6 <<>> -t caa google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27765
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                    IN      CAA

;; ANSWER SECTION:
google.com.             21600   IN      CAA     0 issue "pki.goog"
```

Stopping Malicious Certs: **CT**

Certificate Transparency

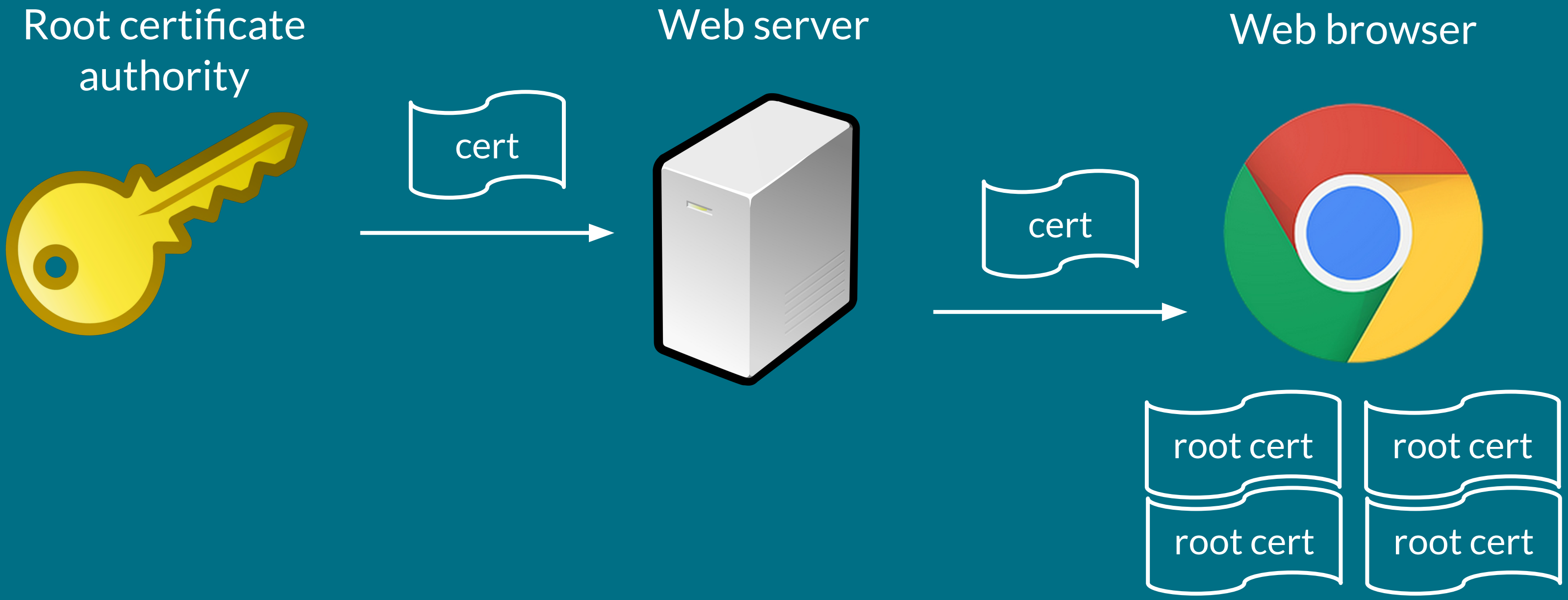
CT: **High-level idea**

Maybe we can't prevent attackers from **getting** a malicious cert,
but maybe we can **detect** those bad certs.

This is more helpful than it seems!

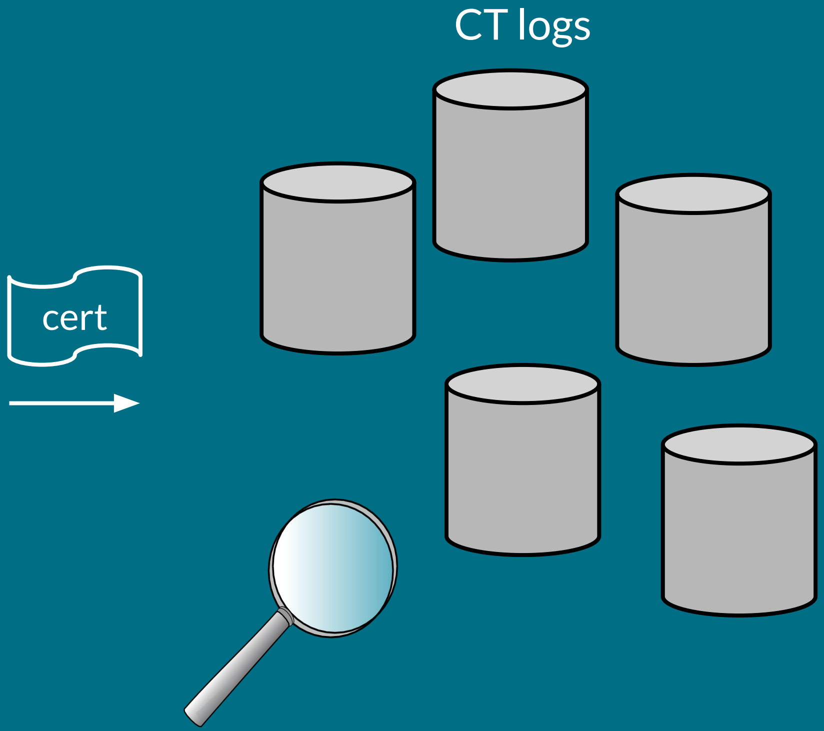
- Makes attacks **noisy**, making them harder to pull off!
- Also helps identify CA problems, so we can fix them!

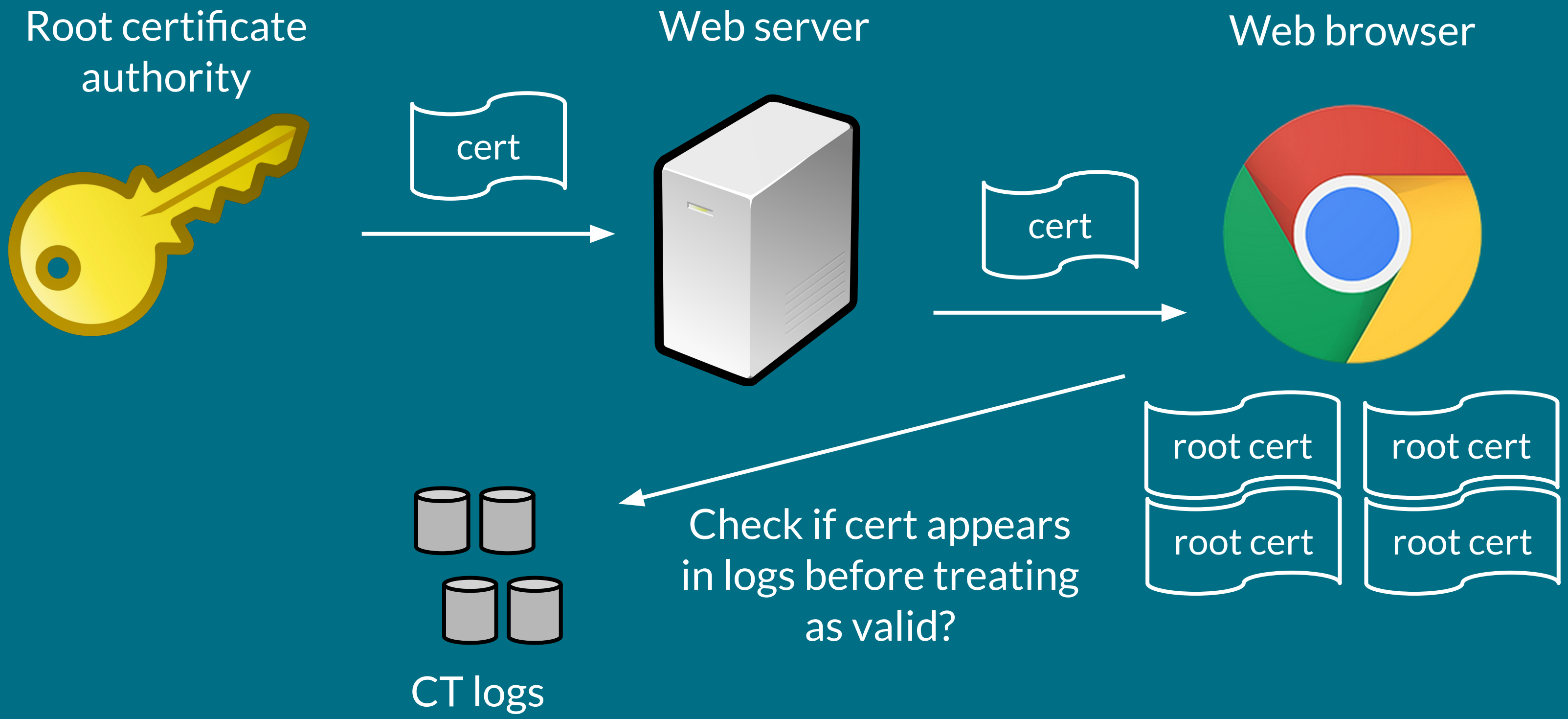
CT: **The Before Times**

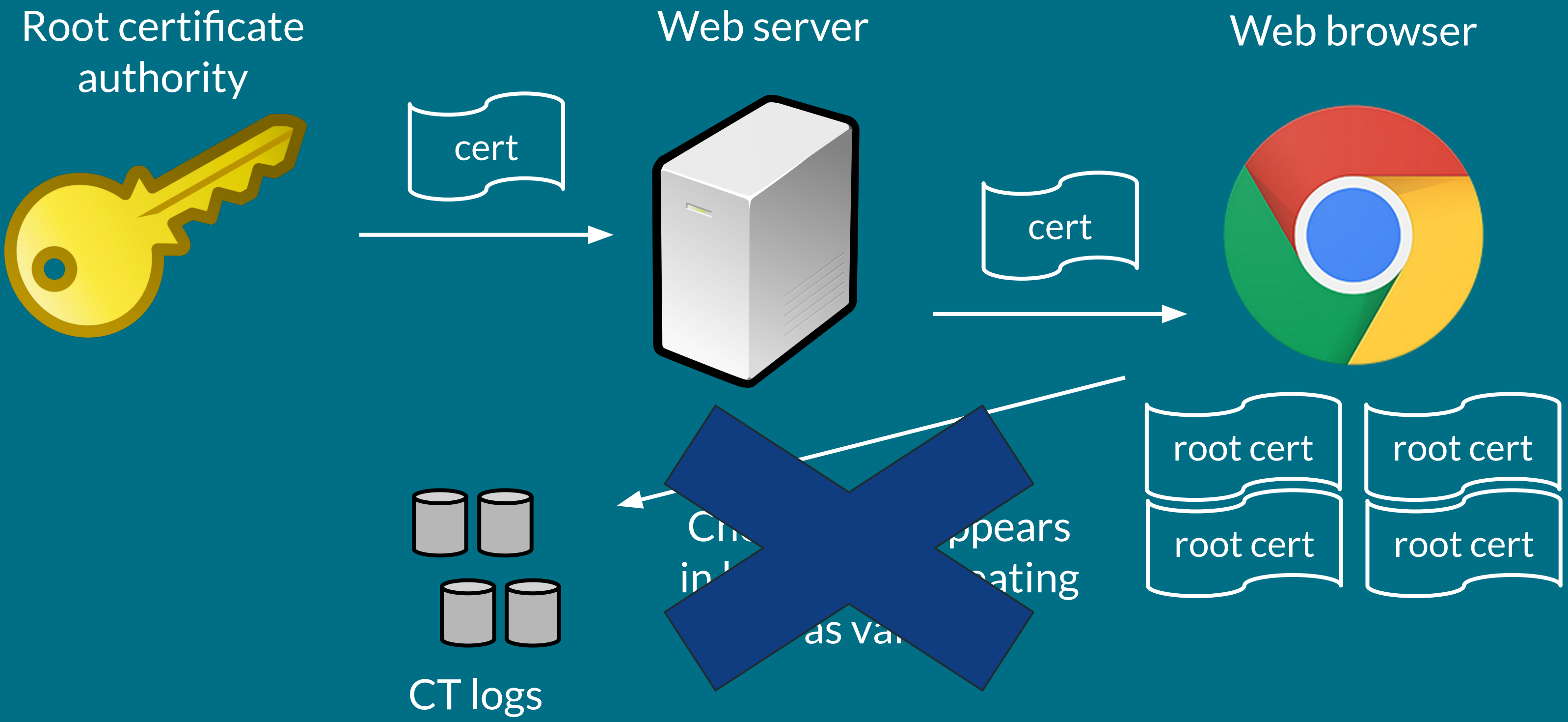


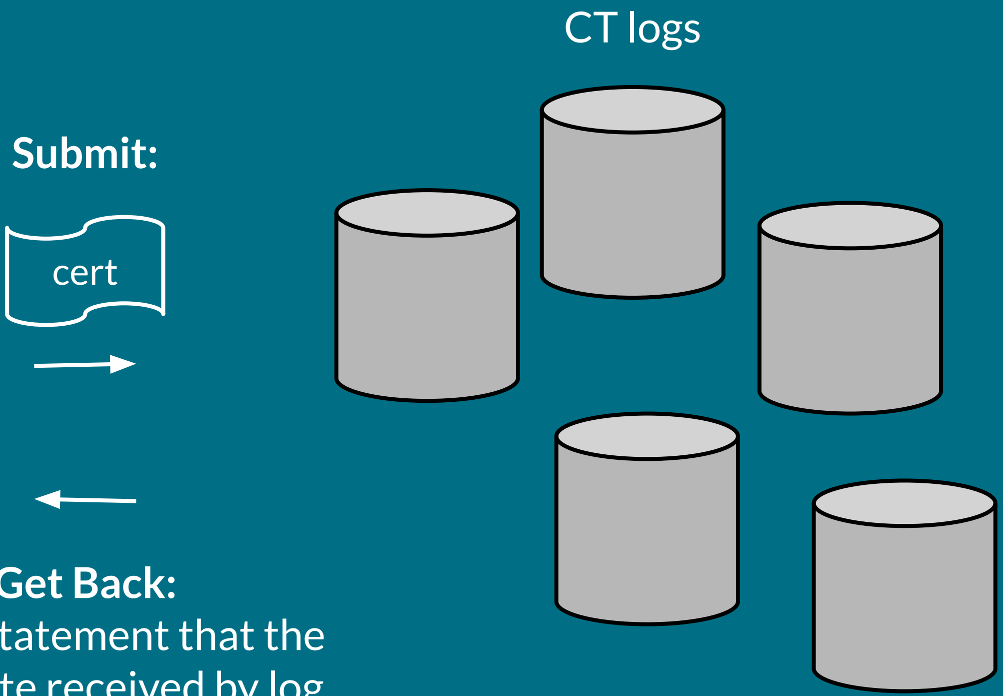
CT: **Public logs**

1. Certificates submitted to public logs
2. Monitors watch logs for malicious certificates

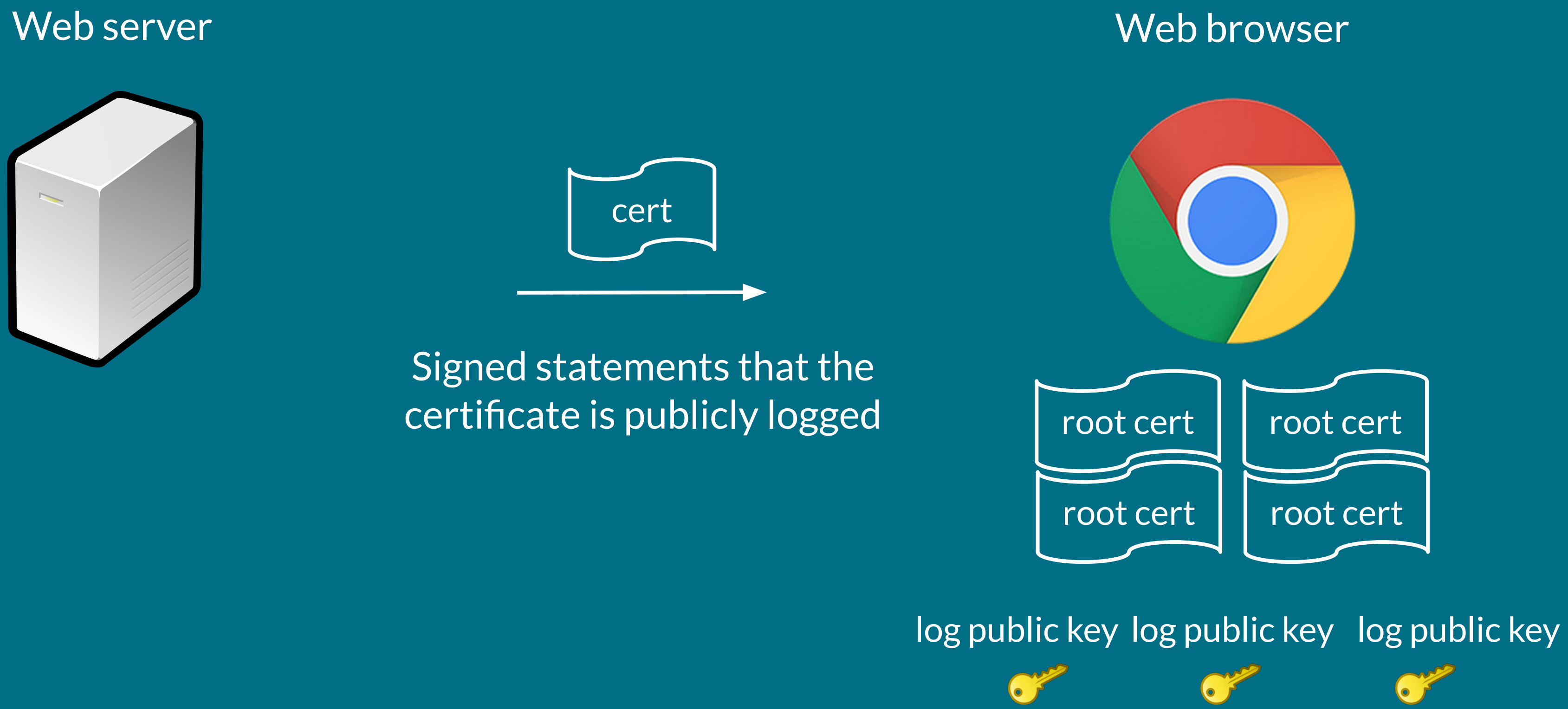



Root certificate authority
cert
Web server
cert
Web browser
root cert
root cert
root cert
root cert
Check if cert appears in logs before treating as valid?
CT logs


Root certificate authority
Web server
Web browser
cert
cert
root cert
root cert
root cert
root cert
Ch
in l
ppears
ating
as va
CT logs



Signed statement that the certificate received by log



Signed statements that the certificate is publicly logged

Web server

Signed statements that the certificate **will be** publicly logged (Signed Certificate Timestamp)

cert

Signe at the
certifica ly logged

Web browser

root cert

root cert

root cert

root cert

log public key

log public key

log public key

CT: **One more detail...**

The logs should be **untrusted.**

Logs might

- say a cert was logged when it wasn't
- give different data to different people

CT: **One more detail...**

The logs should be **untrusted.**

Logs might

- say a cert was logged when it wasn't
- give different data to different people

$\longrightarrow$

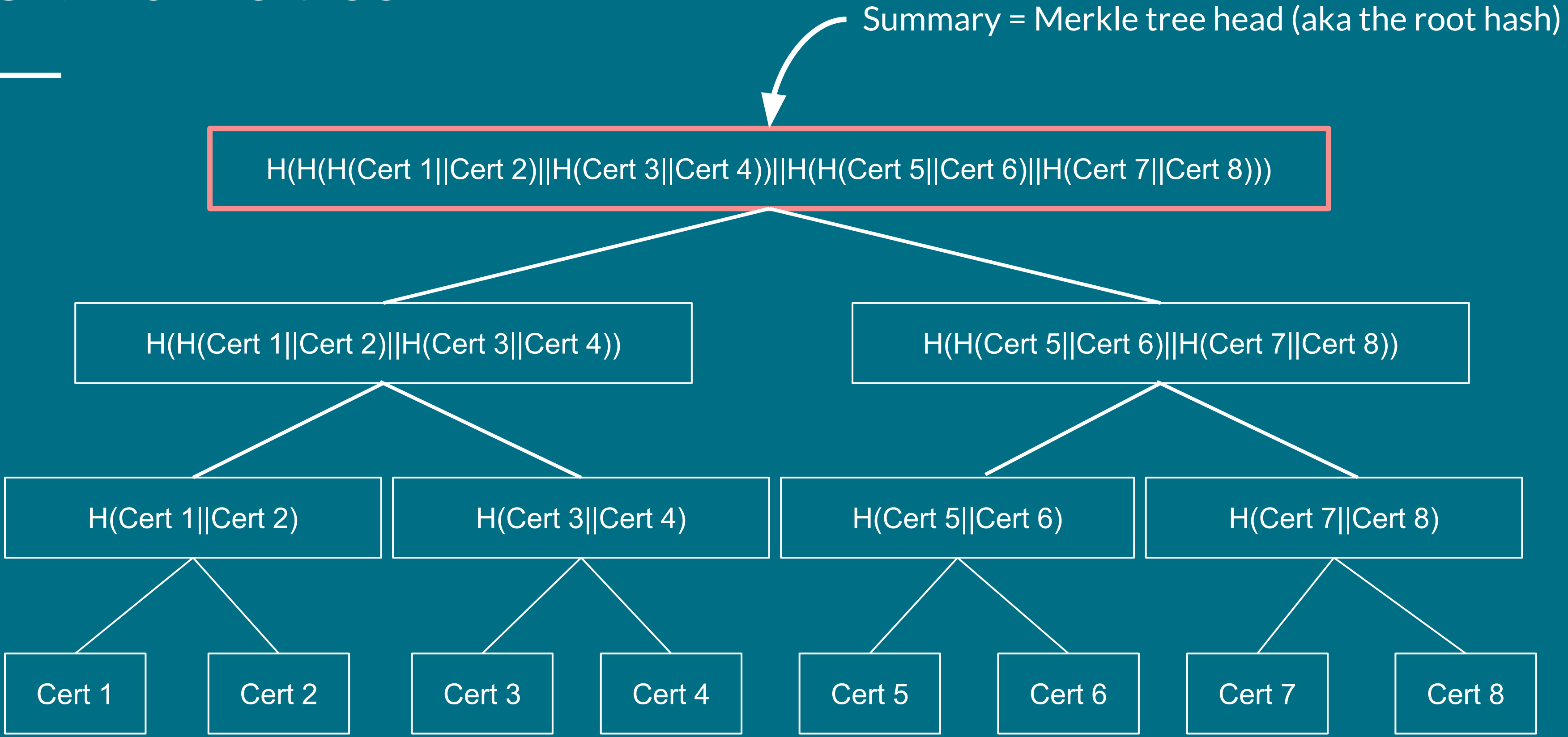
Need a “summary” of log contents.

Lets observers verify

- that a given cert is included,
- that everyone saw the same data.

And *efficiently.*

CT: **Merkle tree**



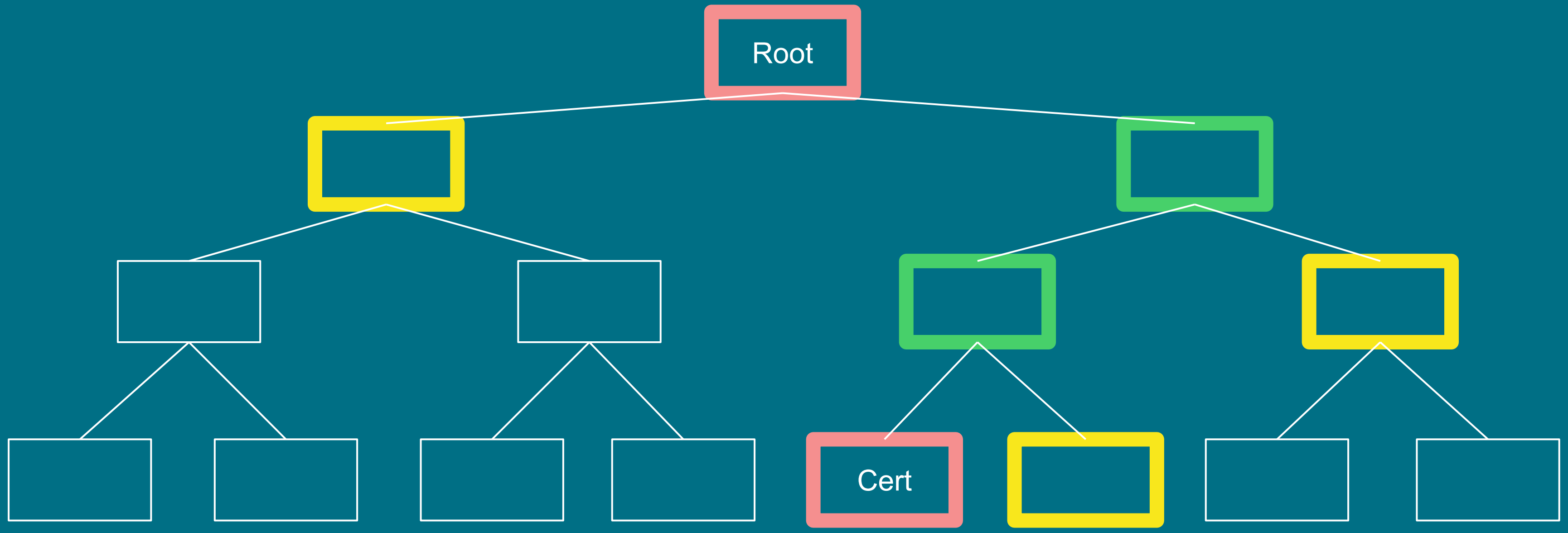
Summary = Merkle tree head (aka the root hash)
H(H(H(Cert 1||Cert 2)||H(Cert 3||Cert 4))||H(H(Cert 5||Cert 6)||H(Cert 7||Cert 8)))
H(H(Cert 1||Cert 2)||H(Cert 3||Cert 4))
H(H(Cert 5||Cert 6)||H(Cert 7||Cert 8))
H(Cert 1||Cert 2)
H(Cert 3||Cert 4)
H(Cert 5||Cert 6)
H(Cert 7||Cert 8)
Cert 1
Cert 2
Cert 3
Cert 4
Cert 5
Cert 6
Cert 7
Cert 8

CT: **Merkle tree properties**

- Only one sequence of certs produces a given root hash
- If two observers calculate the same hash, then they saw all the same certs

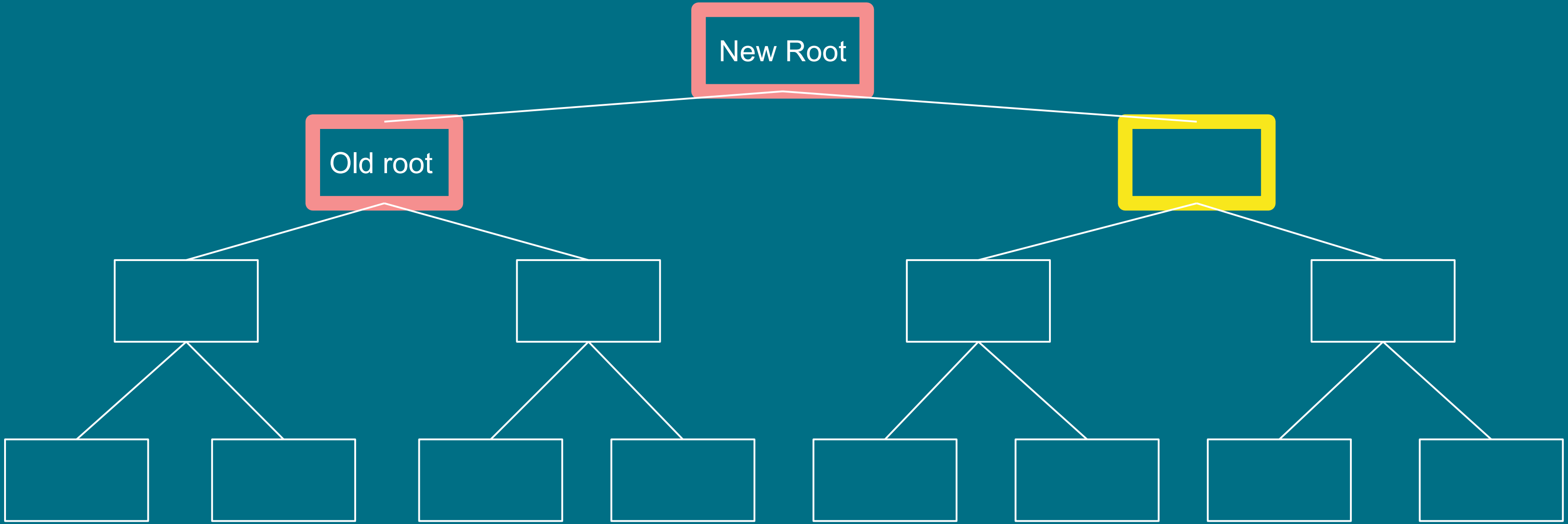
CT: **Merkle tree properties**

To prove that a given cert is included in a root hash, only need log(N) hash values.



CT: **Merkle tree properties**

Similarly, easy to prove that a new root hash is a superset of an old one.



CT: **verifying log honesty**

Observer

1. finds an SCT,
2. gets proof that a cert is included in a root hash,
3. gets proof that new hash includes an older hash, and
4. compares root with others to make sure everyone agrees.

CT: **Promises**

CT **does not prevent** attacks directly

- Attacker can obtain malicious cert and it might not show up in logs for 24hrs
- Maybe longer until observers notice something is wrong with the log

CT offers **detection**

- Good chance that a malicious cert will be detected eventually

CT helps WebPKI **hygiene**

- Helps organizations and researchers discover bad practices

CT: **Organizational hygiene**

“Earlier this year, our Certificate Transparency monitoring service alerted us to an important opportunity to better align internal certificate policies. Specifically, we learned that the Let's Encrypt CA issued two TLS certificates for multiple fb.com subdomains... We determined that **these certificates were requested by the hosting vendor managing these domains for several of our microsites.**”

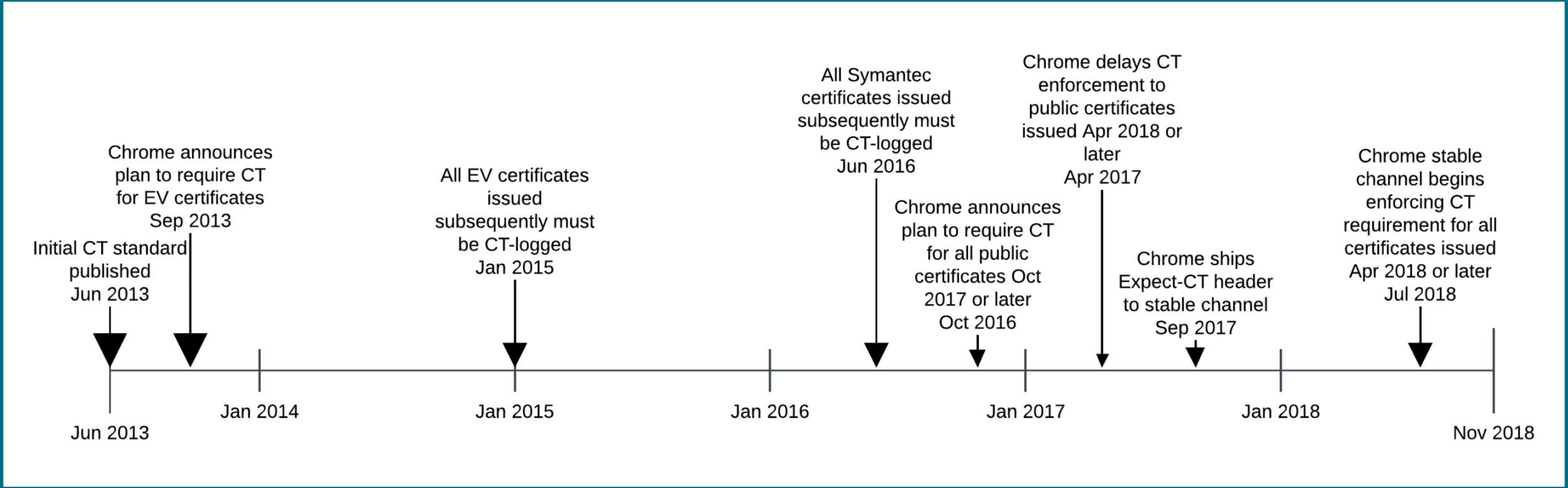
- “Early Impacts of Certificate Transparency” (Facebook, April 2016)

CT: **CA hygiene**

“On September 14, around 19:20 GMT, Symantec’s Thawte-branded CA issued an Extended Validation (EV) pre-certificate for the domains google.com and www.google.com. This pre-certificate was neither requested nor authorized by Google... We discovered this issuance via Certificate Transparency logs... **the issuance occurred during a Symantec-internal testing process**”

- “Improved Digital Certificate Security” (Google, September 2015)

CT: **a work in progress**


All Symantec certificates issued subsequently must be CT-logged Jun 2016
Chrome delays CT enforcement to public certificates issued Apr 2018 or later Apr 2017
Chrome announces plan to require CT for EV certificates Sep 2013
All EV certificates issued subsequently must be CT-logged Jan 2015
Chrome announces plan to require CT for all public certificates Oct 2017 or later Oct 2016
Chrome stable channel begins enforcing CT requirement for all certificates issued Apr 2018 or later Jul 2018
Initial CT standard published Jun 2013
Chrome ships Expect-CT header to stable channel Sep 2017
Jun 2013
Jan 2014
Jan 2015
Jan 2016
Jan 2017
Jan 2018
Nov 2018

CT: **Current state**

Chrome and Safari require and verify SCTs on all certificates.

Chrome *newly* checks that SCTs are included in logs ("SCT Auditing").

- List of visited SCTs ~= list of sites you visited. Hard to share!

But...

- No one checks that logs are presenting consistent views
- These systems are still being designed, built, and deployed!

many open problems.
no easy answers.

Simple solutions, but still open problems

How do we always connect to sites securely?

- How do we fix tracking in HSTS, without sacrificing security?
- What’s the long-term plan for HSTS preloading and static pinning?

How do we ensure that a stolen certificate isn't game-over?

- How can we stop attackers from using stolen certs (HPKP) without the pitfalls?
- How can we verify log honesty in Certificate Transparency?

Many more we didn't talk about...

Questions?