

HTTPS in the real world: a spooky tale

Emily Stark and Chris Palmer
estark@, palmer@chromium.org
@estark37, @fugueish



Introduction

Who are we?

Emily leads the Chrome usable security team:

- Building, maintaining, improving security warnings
- HTTPS adoption and ecosystem improvements
- Site identity and connection security displays
- Research/experimentation around how people use Chrome's security features
- Consulting with other Chrome teams that ask users to make security decisions

(Stanford '11 alum and former CS 253 student :)

Who are we?

Chris leads the Chrome Platform Security team:

- Privilege reduction and privilege separation (“sandboxing”)
- Inter-process communication
- Memory (un)safety, languages and libraries, exploit mitigations

And the Chrome Security Review team:

- Making sure new Chrome (and sometimes web) features are safe enough

Reminder: HTTPS

- https:// provides confidentiality, integrity, authentication.
- But:
 - not all web traffic is https://
 - browsers still support http://
 - https:// is only as strong as the underlying certificate
- Today we'll talk about systems on top of https:// that attempt to solve these problems, with lessons about what worked in the real world and what didn't

Strict Transport Security

Problem: http:// still exists

- Type “example.com” => http://example.com
- Follow http:// link on the web
- Webpage loads http:// subresource

These http:// requests may redirect to https://, but the request may not even make it to the legitimate server; attacker can intercept and tamper.

http:// interception is not theoretical

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.



The screenshot shows a web page from Netresec, a security blog. The header includes the Netresec logo and navigation links: NETRESEC, Products, Training, Resources, Blog, and About Netresec. The main content area displays a tweet from GitHub Status (@githubstatus) dated Tuesday, 31 March 2015 01:15:00 (UTC/GMT). The tweet text reads: "We are working to mitigate an ongoing and evolving large DDoS attack." Below the text are engagement metrics: 72 retweets and 34 favorites, along with a row of user avatars. The tweet is timestamped "9:18 AM - 28 Mar 2015".

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](#)

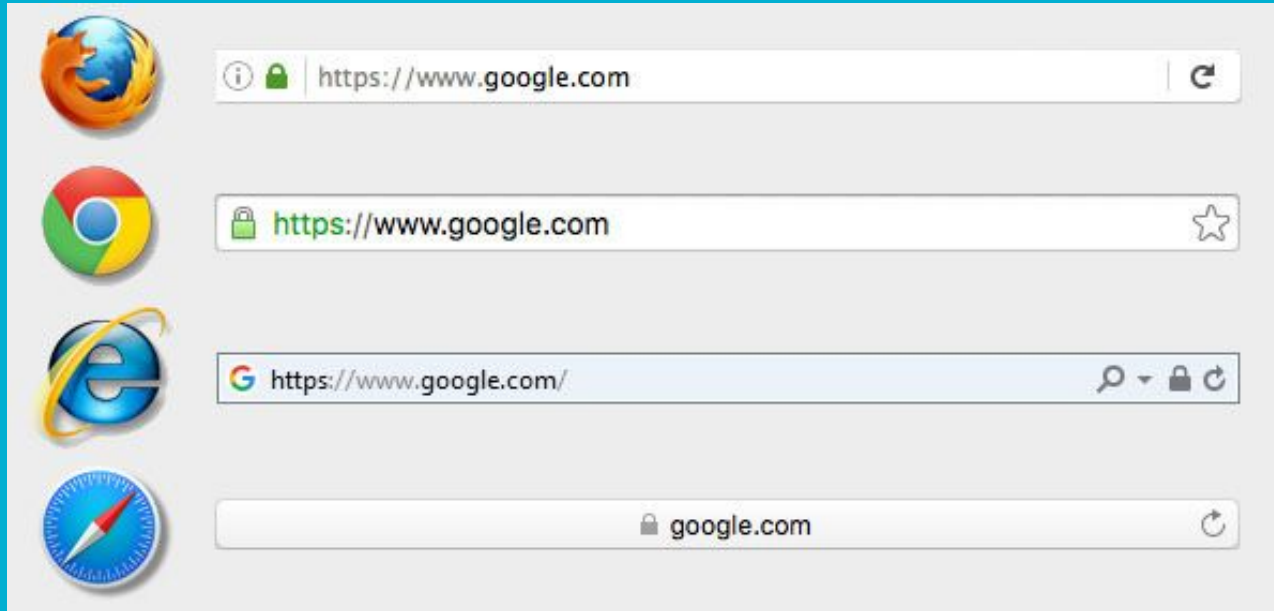
 **GitHub Status**
@githubstatus [Follow](#)

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

RETWEETS 72 FAVORITES 34

9:18 AM - 28 Mar 2015

First defense attempt: positive UI



First defense attempt: positive UI

Not reliable in practice; people don't notice the absence of a security indicator in an attack, and often misunderstand their meaning in benign situations.

- “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.)



Strict Transport Security

Website opts in to strict mode: “I only ever want to be contacted via a secure connection.”

Strict Transport Security

Website opts in to strict mode: “I only ever want to be contacted via a secure connection.”

- If example.com opts into STS, the browser rewrites all requests to `http://example.com` to `https://example.com`.
- If the browser ever observes an invalid certificate on `https://example.com`, it doesn't offer the user the option to bypass the error.



Elements

Console

Sources

Network





Performance

Memory

Application

Security



Audits



☒ Preserve log

☐ Disable cache

Online



500 ms

1000 ms

1500 ms

2000 ms

2500 ms

3000 ms

3500 ms

4000 ms


4500 ms


5000 ms


5500 ms


6000 ms


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

×

Headers

Preview

Response

Timing

▼ 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



Your connection is not private

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

Hide advanced

Reload

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.

Opt in to STS 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`

Opt in to STS 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

Time in seconds for which browser
should cache this STS information

Opt in to STS 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`

Apply STS to all subdomains, e.g. if observed on example.com, STS applied to www.example.com, foo.bar.example.com, etc.

Opt in to STS 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`

To be discussed later...

HSTS gotchas

No way to set `includeSubdomains` for all-but-one-or-two subdomains

- Problematic for large organizations with many subdomains operated by different entities

HSTS gotchas

No standardized way to set HSTS broader than your subdomain

- E.g., when users land on `www.example.com`, no standard way to set HSTS that applies to all of `example.com`
- In practice, `www.example.com` loads an invisible pixel from `example.com`, which sets HSTS for the whole domain

HSTS gotchas

No way to undo HSTS besides waiting

- In practice, organizations deploy with gradually increasing max - age

HSTS gotchas

Does not protect first visit

- To be discussed later...

Big HSTS gotcha: HSTS tracking

- HSTS allows websites to set persistent state which can be set and read in a third-party context
- Persistent state which can be set and read in a third-party context can be abused to track users (like a cookie)
 - Can't be viewed, restricted, or cleared in the same way as cookies

HSTS tracking: how does it work?

Setting the cookie:

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

HSTS tracking: how does it work?

Reading the cookie:

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

HSTS tracking in theory

“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>

HSTS tracking in theory



HSTS tracking in the wild

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

- [“Protecting Against HSTS Abuse”](#) (WebKit blog, March 2018)

Mitigating HSTS tracking

- No perfect solution: all solutions require some trade-off of security and privacy

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 visiting foo.bar.example.com, subresources can only set HSTS if they are loaded from foo.bar.example.com or example.com, not from 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 for a domain, ignore HSTS state for that domain when loading subresources from it
 - Relies on existing complex 3rd party cookie blocking logic

Mitigating HSTS tracking

Chrome's planned mitigations (highly tentative):

- Forcibly autoupgrade all mixed content (http:// subresources on https:// pages)
 - A good idea regardless of HSTS tracking
- Do not apply HSTS upgrades to subresources
 - When mixed content is autoupgraded, applying HSTS provides no value for subresources on https:// pages
 - Minimal security loss from disregarding HSTS for subresources on http:// pages

One more big HSTS gotcha...

Doesn't protect first visit

HSTS preload list

Browsers ship baked-in lists of HSTS sites which are protected on first visit

```
{ "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 },
```

HSTS preload list

Operational nightmare:

- List maintained by Chromium, pulled into other browsers with their policies applied
- Getting off the list requires waiting at least a Chrome release cycle, plus waiting for updates to be pulled into other browsers
- List size grows with HTTPS adoption on the web

HSTS preload list

Some automation:

- Website owners can submit their site to the preload list at hstspreload.org
- Site is checked automatically for inclusion; must serve HSTS header with `preload` and `includeSubdomains` directives, and `max-age` at least 1 year
 - `preload` directive prevents websites from being added against their will
 - `includeSubdomains` directive limits list of eligible sites
- Sites can also be checked automatically for removal
 - `preload` directive must be absent

But we occasionally have weird one-off requests that are handled manually

How will we ever get rid of the preload list?

- Move all websites to https:// and deprecate http://?
- Assume all websites are https:// and show warnings before falling back to http://?
- Somehow define “high value” sites and limit list to those sites?
- Fetch portions of list on demand?
- ...

Key Pinning

Or, “I deleted my Noogler
project. You’re welcome”

Problem: any CA can issue for any origin

This is good: website operators have supplier diversity.

This is bad: attackers have supplier diversity.





Final Report on DigiNotar Hack Shows Total Compromise of CA Servers



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

‘Solution’: borders and boundaries

E.g., only let CAs from country X issue to website operators based in country X.

But recall that there was nothing specific to DigiNotar, the Netherlands, Iran, or Google about the DigiNotar hack. (For more, see [sslmate.com/certspotter/failures.](https://sslmate.com/certspotter/failures/))

Also, the web is world-wide. We want to be able to talk to everyone, and work with everyone!

Also, how would you enforce it?

‘Solution’: pin keys in the client

Similar to SSH: Warn if the server ever presents a different key hierarchy.

This provides 3 things:

- Smooth bootstrapping from the Web PKI and trusted CAs.
- Website operators retain supplier diversity.
- Attackers lose supplier diversity.

Note: We pin **keys**, not **certificates**, and not any non-key aspect of the certificate!!

Example

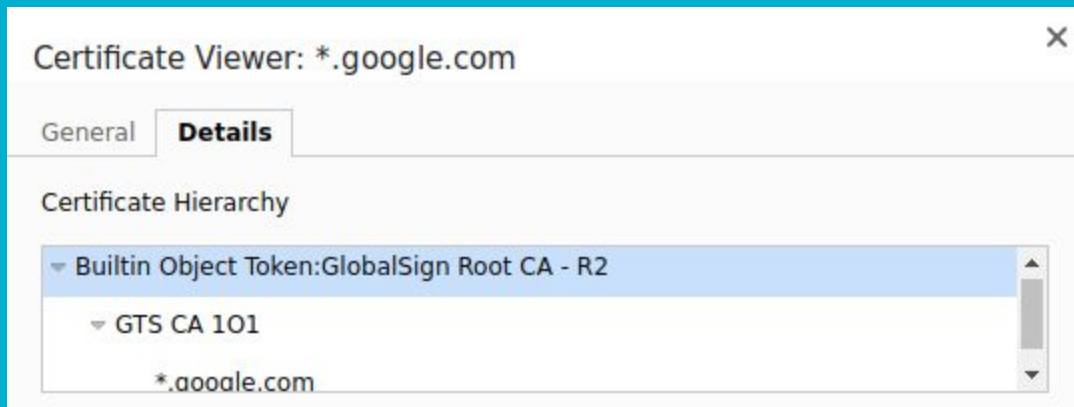
The server sends an HTTP response header describing its **pin set**:

```
Public-Key-Pins: max-age=3000;  
pin-sha256="d6qzRu9z0ECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM=";  
pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g="
```

These are SHA256(`certificate.subjectPublicKeyInfo`).sPKI is a field in the certificate that specifies the key type and the public key bytes.

Pin to any key(s) in the certificate chain

The operator can pin to any 1 or more of these certs' keys. **Pin validation** is set intersection: if any 1 pinned key matches the keys in the validated chain, ✓.



Problem: hard for operators to use

Very few people understand the certificate chain hierarchy, issuer ecosystem, and client behavior.

The chain **as served** \neq the chain **as constructed for validation**.

Operators have no reliable way of knowing what chain the client will validate!

This makes it too unreliable to be a critical Open Web Platform interface.

Problem: hard for browser users to use

Failure is non-recoverable and non-actionable. “Oops”

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.

Problem: hostile pinning

If the unthinkable — server compromise! — happens, the attacker could set a new pin set for their own server. The attacker could:

- Lock clients into the attacker's server
- Deny service to the site, long-term

In many cases, though, server compromise enables so many extreme losses that hostile pinning pales in comparison.

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.

Alternatives to HPKP: CAA

- Advisory — enforced at ‘layer 8’
 - But indeed it has been! 👍
- Enforced after the fact
- Not-yet-secure DNS is the transport

DNS Certification Authority Authorization (CAA) Resource Record

Abstract

The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify one or more Certification Authorities (CAs) authorized to issue certificates for that domain. CAA Resource Records allow a public Certification Authority to implement additional controls to reduce the risk of unintended certificate mis-issue. This document defines the syntax of the CAA record and rules for processing CAA records by certificate issuers.

Alternatives to HPKP: static pinning

Instead of using an HTTP response header discovered and interpreted dynamically, why not bake pin sets into the client?

Because it's a major pain in the ass, that's why.

We do it anyway.

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

Weakness: Same.

Alternatives to HPKP: CT

Emily will explain!

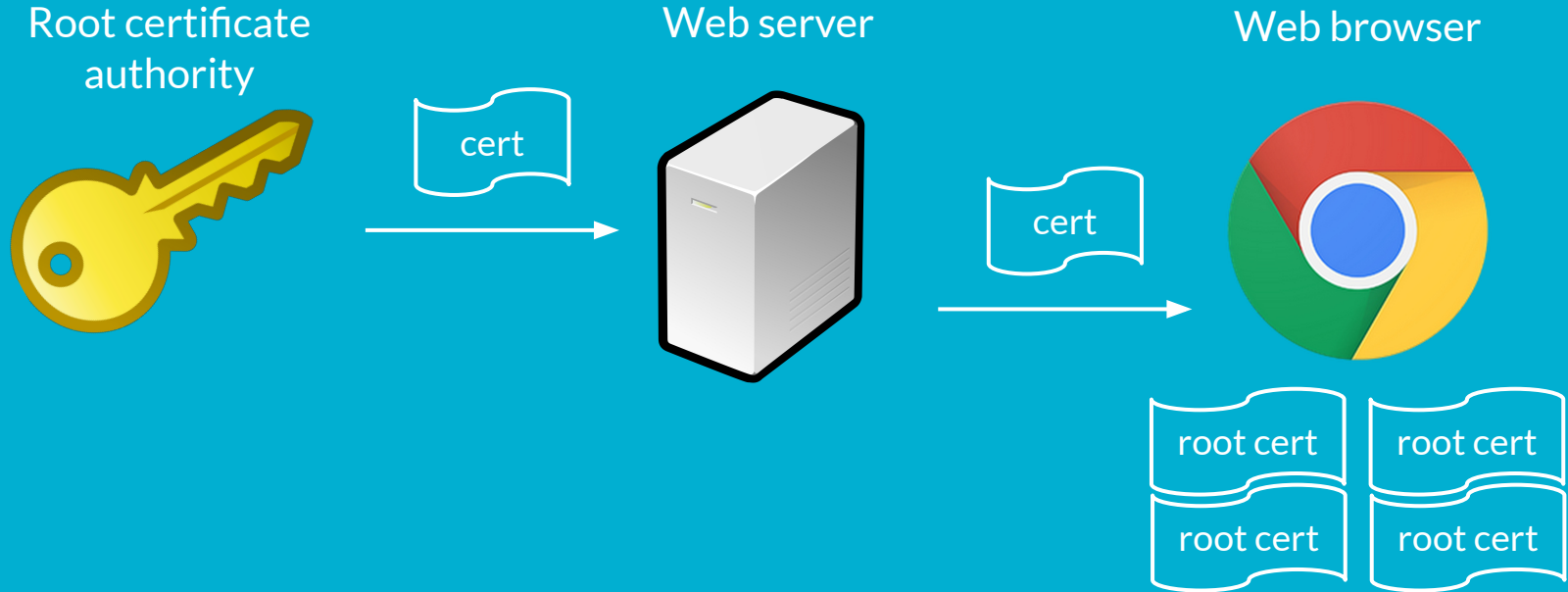
Certificate Transparency

High-level idea

We may not be able to prevent attackers from using malicious certificates, but we may be able to make all certificates **discoverable**.

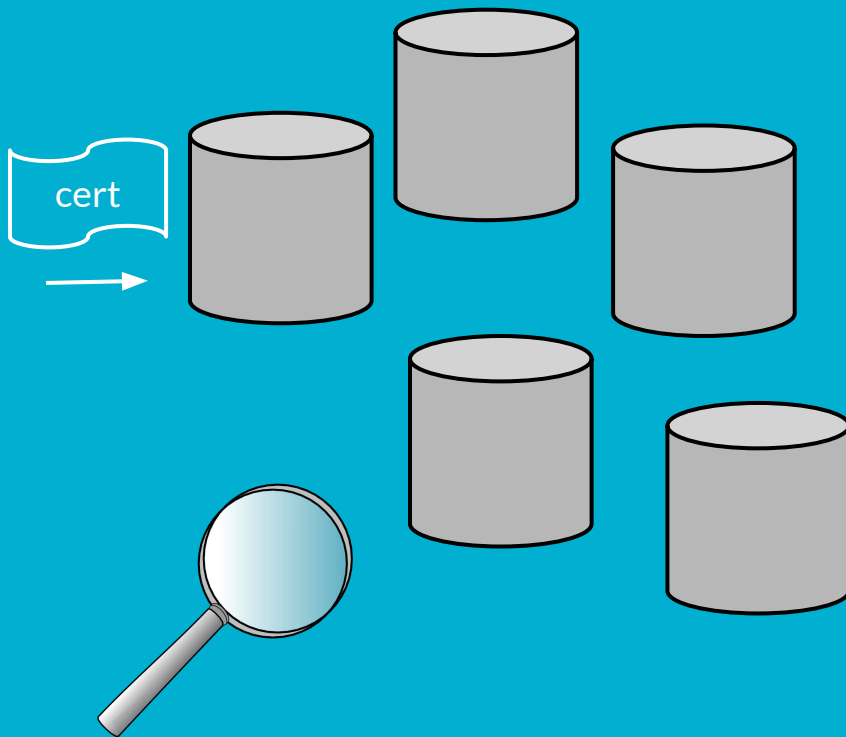
- Discourage malicious issuance
- Reveal accidental issuance
- Make attacks noisy

Before CT

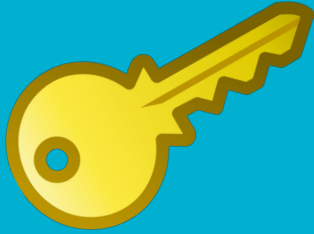


Public logs

- Certificates go into public logs
- Monitors can watch logs for malicious certificates



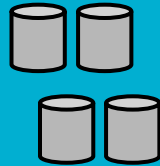
Root certificate
authority



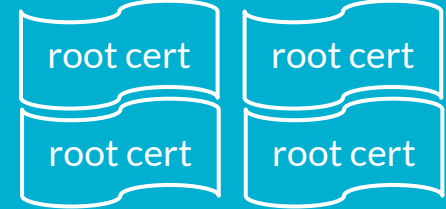
Web server



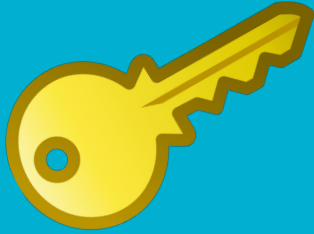
Web browser



Check if cert appears
in logs before treating
as valid?



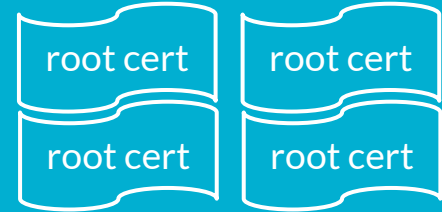
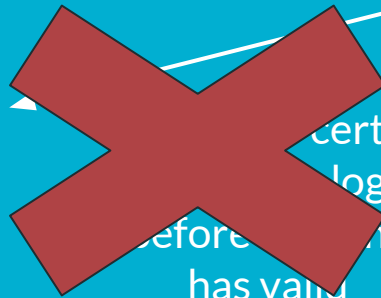
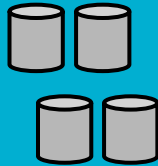
Root certificate authority



Web server



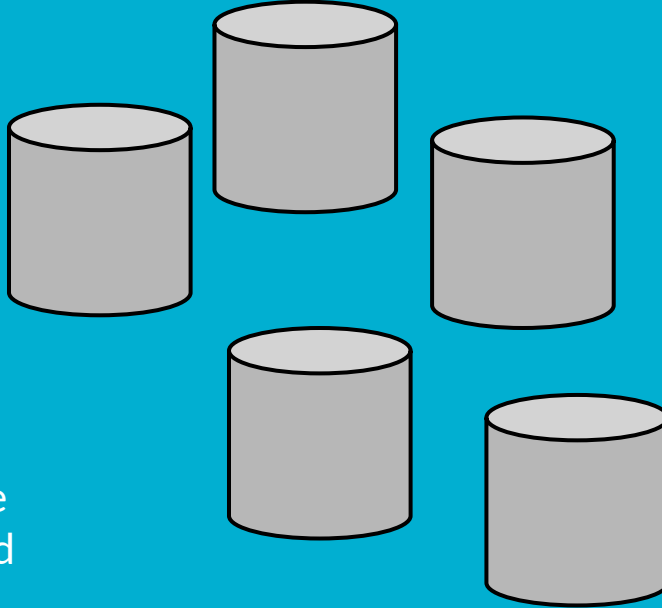
Web browser



cert
logs
before
ing
has valid



Signed statements that the
certificate is publicly logged

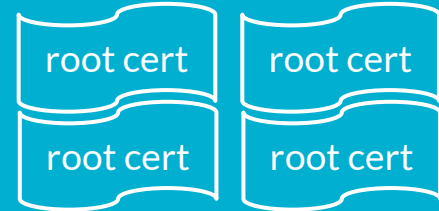


Web server



Signed statements that the
certificate is publicly logged

Web browser



log public key log public key log public key



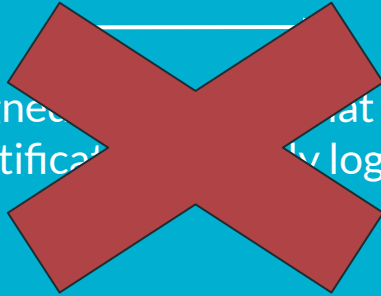
Web server



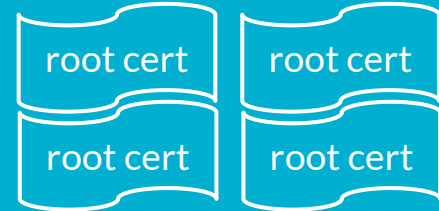
Signed statements that the
certificate **will be** publicly logged
(Signed Certificate Timestamp)



Signed statements that the
certificate **will be** publicly logged



Web browser



log public key log public key log public key



One more challenge...

The logs should be untrusted

- A log might lie and say that it logged a certificate when it didn't
- A log might present different data to different observers


How to detect misbehaving logs?

Need a “summary” of the logged certificates that allows:

- Two observers can compare summaries to efficiently verify that the log has given them the same data
- An observer can efficiently verify that a given cert is included in the summary

Merkle tree

Merkle tree head (the “summary”)



$H(H(H(\text{Cert } 1 || \text{Cert } 2) || H(\text{Cert } 3 || \text{Cert } 4)) || H(H(\text{Cert } 5 || \text{Cert } 6) || H(\text{Cert } 7 || \text{Cert } 8))))$

$H(H(\text{Cert } 1 || \text{Cert } 2) || H(\text{Cert } 3 || \text{Cert } 4))$

$H(H(\text{Cert } 5 || \text{Cert } 6) || H(\text{Cert } 7 || \text{Cert } 8))$

$H(\text{Cert } 1 || \text{Cert } 2)$

$H(\text{Cert } 3 || \text{Cert } 4)$

$H(\text{Cert } 5 || \text{Cert } 6)$

$H(\text{Cert } 7 || \text{Cert } 8)$

Cert 1

Cert 2

Cert 3

Cert 4

Cert 5

Cert 6

Cert 7

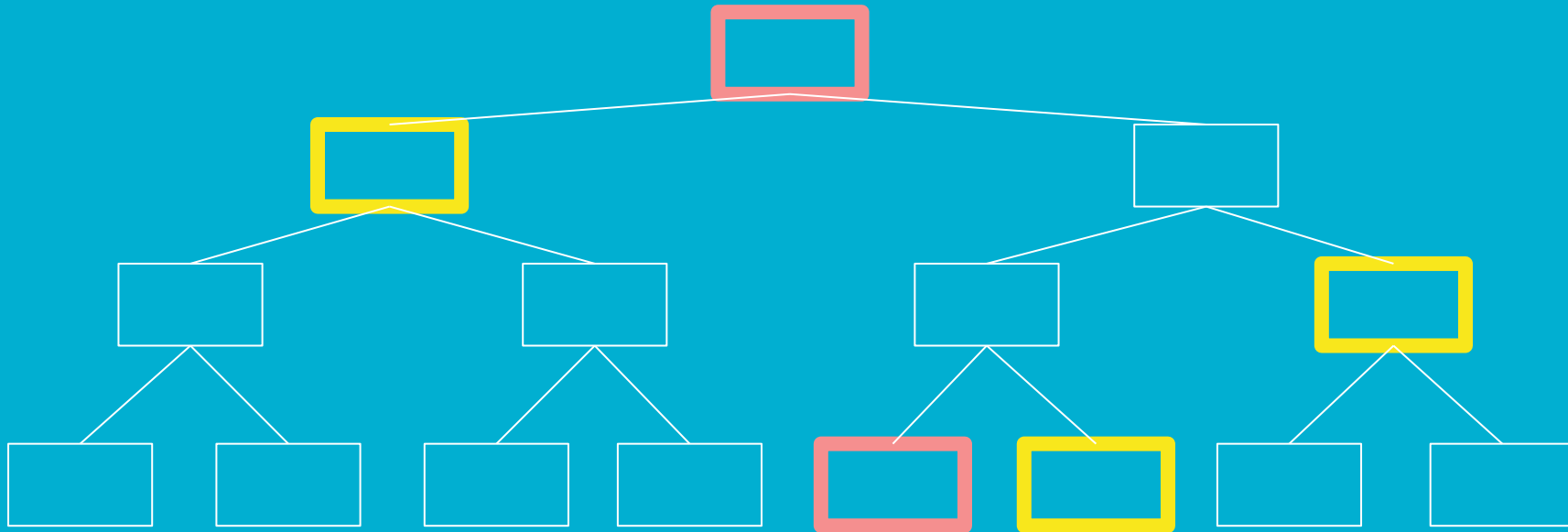
Cert 8

Merkle tree properties

- You can only find one sequence of certificates that produces a given summary
- If two observers have equal summaries, then they saw the same sequence of certificates

Merkle tree properties

To prove that a given cert is included in a “summary”, only need $\log(N)$ hash values.



Merkle tree properties

Similarly, small # of hashes needed to prove that a new summary is a superset of an old one.

Using Merkle trees to verify log honesty

- When an observer sees an SCT (promise from a log to include a certificate), it can ask for a log summary with a proof that it includes that certificate
- Observers can efficiently compare log summaries and verify that their observed summaries are consistent with each other

What promises does CT make?

Not prevention:

- Attacker can obtain malicious cert and it might not show up in logs for 24hrs
- Unspecified delay until observers might notice inconsistencies in log behavior

Detection:

- Good chance that a malicious cert will be detected eventually

Hygiene:

- Helps organizations and researchers discover bad practices

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.**”

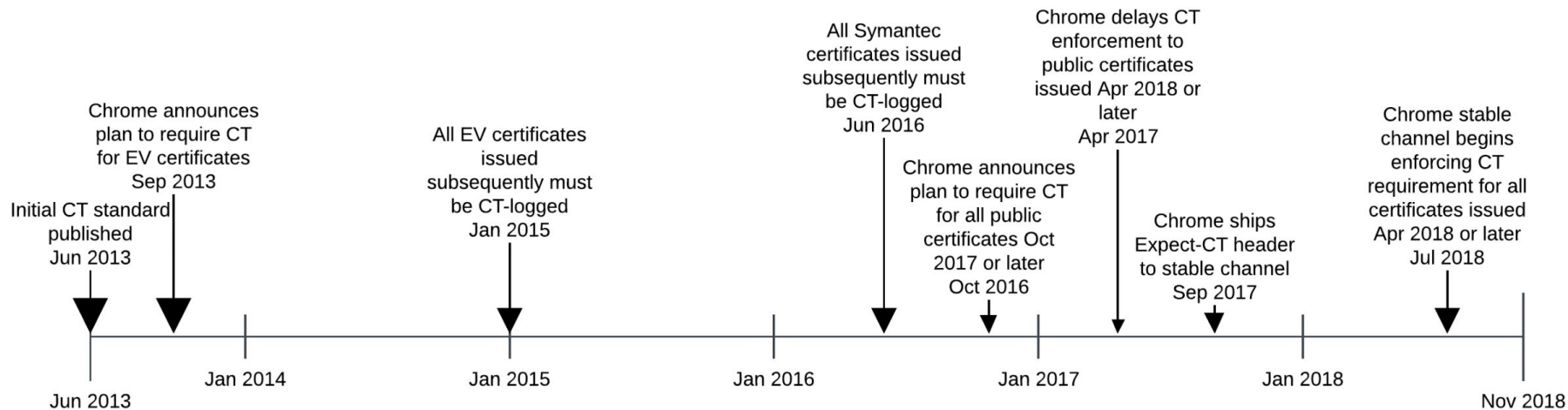
- “Early Impacts of Certificate Transparency” (Facebook, April 2016)

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 deployment: a work in progress



Current state of CT deployment

Chrome requires and verifies SCTs on most certificates

But...

- No one checks that certs served to browsers actually appear in logs
- No one checks that logs are presenting consistent views
- These systems still need to be designed, built, and deployed!

Conclusion: open problems

Open problems

- What's the best way to prevent HSTS tracking without sacrificing too much security?
- What's the long-term plan for the HSTS preload list and static pinning list?
- How can we achieve security guarantees of HPKP without the pitfalls?
- How do we build systems to verify CT log honesty?

Questions?