Certificate Transparency

Adam Langley, Ben Laurie, Emilia Kasper
Al Cutter, Stephen McHenry

{agl, benl, ekasper}@google.com

Part I
Certificate Transparency Background & Design

Part II
Implementing A CT Log

The DigiNotar/TURKTRUST story

- July 19th, 2011: DigiNotar CA finds evidence of compromise through routine daily check
- Evidence of large-scale MitM in July
- *.google.com pinning failure externally reported August 28th, cert revoked and Chrome updated August 29th

---

- August 2011: TURKTRUST CA mistakenly issues two intermediate CA certs
- *.google.com cert detected December 24th 2012, revoked December 25th

How to fix this?

- Minimize the window between incident and response
  - We can't prevent attacks, but we can make them much more expensive by giving the attacker only one, short-lived shot
- Only domain owners know which certificates are legitimate - give them power
- Make the (computers of the) world gossip
  - vaccination effect: not everyone has to participate for everyone to benefit

Certificate Transparency Promise

Certificate Transparency will make all public end-entity TLS certificates public knowledge, and will hold CAs publicly accountable for all certificates they issue.

And it will do so without introducing another trusted third party.

Design requirements

- Compulsory: make non-logged certs hard fail in browsers
  - Must be extremely easy for server operators (= no software upgrade)
  - No side channels (a la OCSP) in TLS handshake
  - No noticeable performance penalty for page load
- Backwards compatible: do not break old browsers
- No plug-and-play option in TLS...
- ... but can do hard fail with CA participation
  - CA submits cert, embeds signature and re-signs

Certificate log core design

- A CT Log is an append-only list of certificates. The log server
  - Verifies the certificate chain for CA attribution
  - For accepted certificates, immediately issues a cryptographic promise to log them
  - Periodically appends all new certificates to the append-only log and signs that list (we use a Merkle Tree)
- Two-phase design influenced by both CA/TLS server and log server deployment restrictions

Who participates in the protocol?

- Server(operator)s and CAs
  - submit certificates to the log
  - obtain a signature that a certificate is logged
  - servers present this signature to TLS clients
- TLS clients
  - synchronously verify the log signature using a built-in public key
  - asynchronously verify that the certificate has appeared in the append-only log
  - asynchronously gossip their view of the log
- Everyone
  - verifies their views of the log are consistent
  - monitors the log for suspicious certificates

Public reactions

- Lots of supportive reactions:
  - "DigiCert believes strongly in the value of added transparency to [SSL]" (Jeremy Rowley, DigiCert)
  - "FWIW, as lead developer of Comodo's issuance code [...], I intend to seek permission [...] to implement [CT]." (Rob Stradling, Comodo)
  - "I think [CT] lets everyone win without being the TSA of the Internet." (Jon Callas, Entrust)

- Some (valid) concerns from CAs:
  - What if we want to issue a cert and the log is down?
  - What if the log rejects our certificate?

Who will operate logs?

- Google is committed to running a robust, high performance log service
- We hope that there will be other logs
  - Multiple logs = feasible to revoke a compromised log's key in the browser
  - Certs can have multiple log signatures, clients will check that at least one of them is from a currently trusted log
- Not every log has to be high-performance
  - Open-source codebase for smaller logs
- We'd welcome CAs to run one to alleviate concerns about external dependencies

Part II
Implementing A CT Log

System requirements

- Seamlessly integrate with CA processes
  - Distributed frontends/geographically separate logs for speed and ~100% uptime
  - Reliably commit new certificate entries inline with the certificate submission
  - << 1 qps writes on average (a few million new certs per year?) but highly bursty
- Eventually, assign a fixed order to entries (distributed log needs a global counter)
- Log has a Maximum Merge Delay (MMD) for publishing updates

CT log security

- The private key
  - Log key is as sensitive (= as hard to replace) as a root CA key...
  - ... but unlike a root CA key, needs to be online 24/7
  - However ROI on compromise is much smaller:
    - attacker still needs to compromise a CA first
    - ... and only gets one, short-lived shot
  - Need multiple logs to tolerate occasional failure
- Crypto
  - RSA2048/ECDSA P-256 with SHA256
  - ECDSA has minimal overhead in embedded certificates (~100 bytes per CT log)

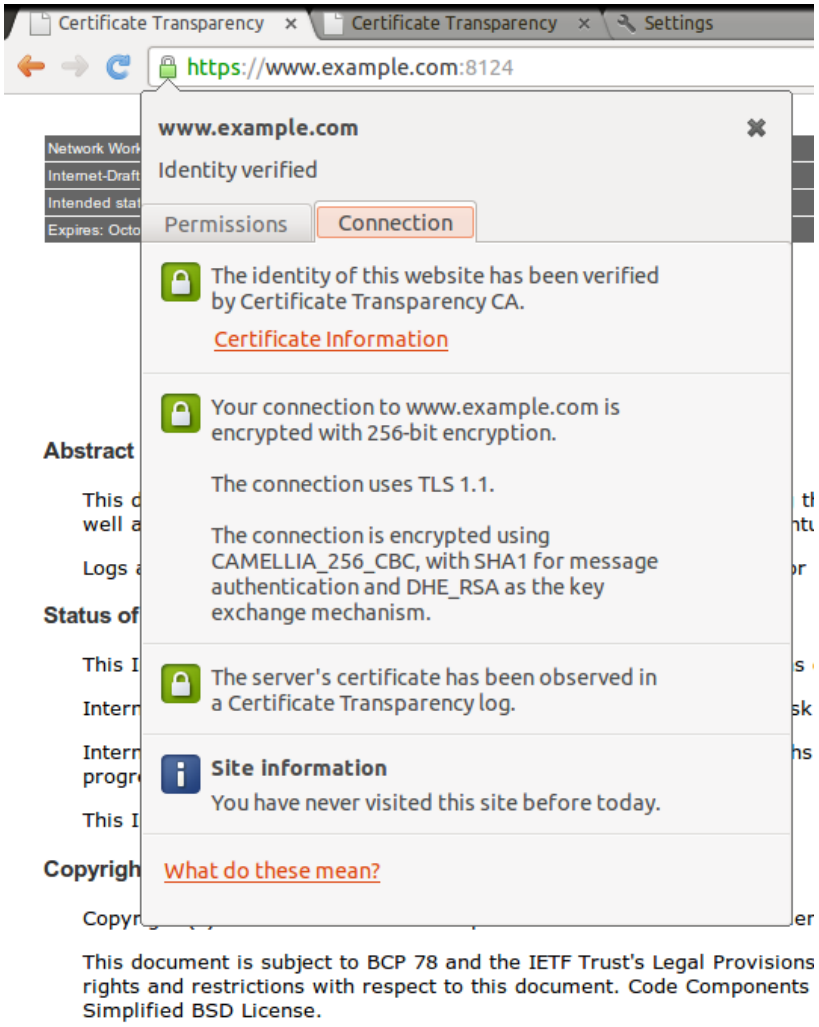
Google is running a CT pilot log

https://ct.googleapis.com/pilot/

- 1,247,715 certificates in the log, and counting (https://ct.googleapis.com/pilot/ct/v1/get-sth)
- No official MMD yet but aim to update daily
- Aim to accept common roots: if yours is missing and you'd like it added, let us know
- Public key and updates via certificate-transparency@googlegroups.com

CT support for Chrome coming soon

- Gradual deployment: start by displaying positive indicators for CT-enabled websites
- Privacy-preserving gossip protocols
- Allow "pinning" of mandatory CT?



Resources

Design document

http://www.links.org/files/CertificateTransparencyVersion2.1a.pdf

Experimental Internet Draft http://datatracker.ietf.org/doc/draft-laurie-pki-sunlight/

Open-source code repository

http://code.google.com/p/certificate-transparency

Google's CT log pilot

https://ct.googleapis.com/pilot

Mailing list

certificate-transparency@googlegroups.com

Q & A

certificate-transparency@googlegroups.com