Certificate Transparency

One Year Milestone

Adam Langley, Ben Laurie, Emilia Kasper

{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
- Backwards compatible: do not break old browsers
- Spent ~6 months analyzing every TLS setting, came up with nothing airtight...
- Need CA involvement for hard fail
  - CA submits cert, embeds signature and re-signs

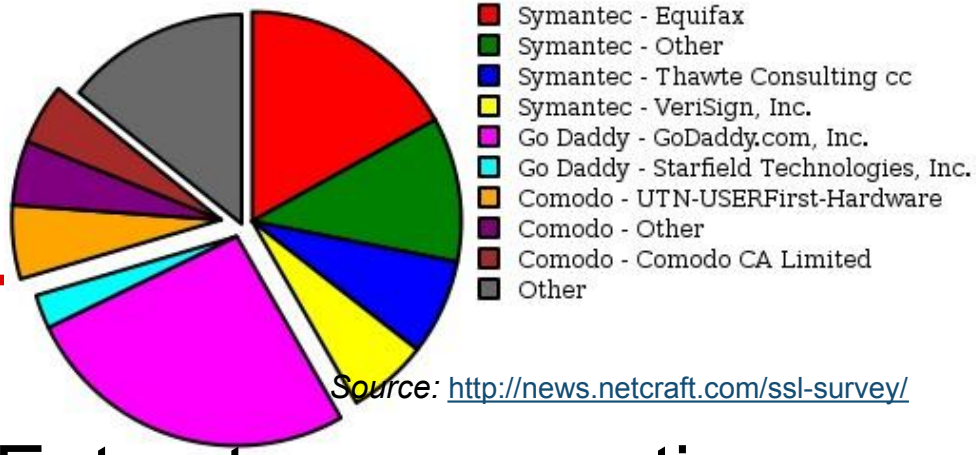
Certificate log core design

- 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

CA reactions



Source: http://news.netcraft.com/ssl-survey/

- DigiCert, Comodo, Entrust are supportive
  - "FWIW, as lead developer of Comodo's issuar code [...], I intend to seek permission from Co Management to implement [CT]." (Rob Stradling)
  - "I can't say enough good things about CT bec think it lets everyone win without being the TS the Internet." (Jon Callas, Entrust)
- Others are more concerned
  - 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
  - Not worth as much to attackers (they still need to compromise a CA first!)
- Crypto
  - RSA 2048+/ECDSA P-256 with SHA256
- User inputs
  - Log has to parse untrusted ASN.1 certificates
  - Open-source implementation uses OpenSSL ASN.1 library: thoroughly tested, few bugs in last 10 years

A single mistake can be fatal...

Mistake indistinguishable from malice. If the log

- signs two conflicting versions of its state; or
- fails to publish updates in a timely manner

then the public has the right to **revoke** its trust in the log. Revocation ≈ root CA revocation (remove trust from browsers). **GAME OVER** for this log.

So how to deal with...?

- Infrastructure failures
  - A log should determine and publicly communicate an SLA - including its Maximum Merge Delay
  - MMD needs to be long enough to allow for manual engineer intervention
- Hardware failures
  - Replicate, replicate, replicate (it's a few GB data)
  - Verify all crypto on the same + a second machine
- Software bugs
  - Can't afford critical bugs...
  - Open-source codebase for core security components

Resources

Design document

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

Experimental Internet Draft in IETF Last Call

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

Open-source code repository

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

Mailing list

certificate-transparency@googlegroups.com

Q & A

{agl, benl, ekasper}@google.com