Certificate Transparency workshop

Eran Messeri, eranm@google.com

This workshop is...

- Oriented towards CAs wanting to support CA using precertificates.
- Probably redundant if you have implemented CT-related functionality.
- Based entirely (and solely) on RFC6962.

Agenda

- Current status of CT deployment.
- Data structures used by CT.
- Submit certificates to logs.
- Generating Precertificates, certs with SCTs.
- Overview of existing code and test data.
- Getting further involved with CT.

CT Deployment - current status

- 6 Logs in operation, recognized by Chrome.
  - Operators: Certly, Digicert, Izenpe, Google (3)
- CT support required by Chrome for EV certs.
- Open-source code implementing RFC6962
  - Including production-quality log server (“super duper”)
- RFC6962-bis in progress (standards-track).

What does “CT Support” mean?

Signed Certificate Timestamps (SCTs) must be presented in SSL connections.

Number varies depending on delivery method.

Methods for delivering SCTs

- Embedded in the certificate.
  - Clients have to strip out!
- During the TLS handshake
- Embedded in stapled OCSP responses.

While embedded SCTs are the focus today, the other methods are more recommended.

What's Signed Certificate Timestamp

- Cryptographically-signed promise from a CT log to incorporate a certificate into the tree.
- Returned upon certificate chain submission
- Contains log ID, timestamp, signature.
- Signature is over timestamp combined with EE cert.

Signed Certificate Timestamp (SCT)

{"sct_version":0,
"id":"
pLkJkLQYWBSHuxOizGdwCjw1mAT5G9+443fNDsgN3BA
=",
"timestamp":1365427532443,
"extensions":"",
"signature":"
BAMARzBFAiEAid6Jf2A+WQsaoNfEI2wvaX6QYCeV96RpI
V/aXkYBI/wCIGWrUBzj269JvVY9HJ/wrHYSC8EfZaRBIrPN
i4n8d6SM"}

Binary encoding of SCTs

- The encoding used for delivering SCTs to clients.
- Based on TLS encoding (RFC5245 section 4).

Structure definition

```
struct {
    Version sct_version;
    LogID id;
    uint64 timestamp;
    CtExtensions extensions;
    digitally-signed struct { …        ← Not the data, signature over it.
     } signed_entry;
    };
} SignedCertificateTimestamp;
```

Structure definition

```
enum { v1(0), (255) } Version;

struct {
  opaque key_id[32];
} LogID;

opaque CtExtensions<0..2^16-1>;
```

Binary-encoded example

‘00a4b90990b418581487bb13a2cc67700a3c3
59804f91bdfb8e377cd0ec80ddc10000000013de9
d2b29b0000040300473045022100 89de897f60
3e590b1aa0d7c4236c2f697e90602795f7a4692
15fda5e460123fc022065ab501ce3dbaf49bd56
3d1c9ff0ac76120bc11f65a44122b3cd8b89fc77
a48c’

Example (con’d)

- Version and timestamp are fixed-size numbers (red).
- LogID is fixed-length opaque strings (blue).
- CtExtensions is variable-length opaque string (in black).
- Signature is hash alg id and sig alg id (fixed-size numbers), then var-length opaque

Submitting a certificate

POST https://<log server>/ct/v1/add-chain

Input encoded in JSON, containing:

chain:

- Array of base64-encoded certificates (in DER form).
- Starting with the end-entity certificate.
- Ending with root cert or a cert chaining to it.

Precertificate - Creation

1. Create TBSCertificate as usual
2. Add poison extension
3. Sign
4. Submit using add-pre-chain
5. Encode collected SCTs to SCTList
6. Add SCTList to TBSCertificate
7. Sign

Poison extension

- Critical X.509 extension
- OID 1.3.6.1.4.1.11129.2.4.3
- extnValue: ASN.1 NULL data encoded in OCTET STRING.

How it looks like in openssl:

1.3.6.1.4.1.11129.2.4.3: critical

..

Precertificate - submission

Same as X.509 certificate submission, using add-pre-chain

Embedding SCTs in Precertificates

```
opaque SerializedSCT<1..2^16-1>;
 struct {
   SerializedSCT sct_list <1..2^16-1>;
 } SignedCertificateTimestampList;
```

Encode as OCTET STRING and add extension with OID 1.3.6.1.4.1.11129.2.4.2.

Validating SCTs - signed data

```
digitally-signed struct {
  Version sct_version;
  SignatureType signature_type = certificate_timestamp;
  uint64 timestamp;
  LogEntryType entry_type;
    select(entry_type) {
      case x509_entry: ASN.1Cert;
      case precert_entry: PreCert;
} signed_entry;
```

Validating SCTs - process

- Serialize all data according to TLS encoding.
- Use the log's public key to verify the signature over the data (code).

Validating SCTs over X.509 certs

```
digitally-signed struct {
  Version sct_version;
  SignatureType signature_type = certificate_timestamp;
  uint64 timestamp;
  LogEntryType entry_type;
    select(entry_type) {
      case x509_entry: ASN.1Cert;
      case precert_entry: PreCert;
} signed_entry;
```

Validating SCTs over X.509 certs

- Serialize all fields.
- Serialize the DER form of the X.509 certificate as variable-length opaque string.

Validating SCTs over Precertificates

```
digitally-signed struct {
  Version sct_version;
  SignatureType signature_type = certificate_timestamp;
  uint64 timestamp;
  LogEntryType entry_type;
    select(entry_type) {
      case x509_entry: ASN.1Cert;
      case precert_entry: PreCert;
} signed_entry;
```

Validating SCTs over Precertificates

```
struct {
  // SHA-256 hash of the issuer's public key
  opaque issuer_key_hash[32];
  // Only the TBSCertificate part of the Precert
  TBSCertificate tbs_certificate;
} PreCert;
```

Final issuer can differ from the precert issuer!

Making sure it actually works

- Test data
- Testtube
- Reference implementations:
  - Java
  - C++
  - Python
- CT in Chrome

Getting further involved

- RFC6962-bis: Feedback from CAs is particularly important.
- Customer/webmaster education.
- Monitoring CT logs.

Filler: Precertificate Signing Cert

- To avoid using the same key for signing the precertificate and end-entity certificate.
- Must be certified by the certificate used for signing the final certificates.
- Identified by OID 1.3.6.1.4.1.11129.2.4.4
- Affects SCT validation.