Certificate Authority Transparency and Auditability

Ben Laurie <benl@google.com>
Adam Langley <agl@google.com>

22 Nov 2011

Goal

The goal is to make it impossible (or at least very difficult) for a Certificate Authority (CA) to issue a certificate for a domain without the knowledge of the owner of that domain. A secondary goal is to protect users as much as possible from mis-issued certificates.

It is also intended that the solution should be backwards compatible with existing browsers and other clients.

Overview

There are various pieces. Firstly, every publicly visible certificate should be published in a **publicly auditable certificate log**. Secondly, each certificate issued must be accompanied by an **audit proof**. Thirdly, servers must send these proofs along with the certificates to browsers, and browsers must check them. Finally, domain owners should monitor the public logs in order to discover if any certificates have been issued that should not be.

In this document we will describe the high-level design - detailed decisions about exact cryptographic algorithms will be deferred at this time.

Trust Model

Browsers will need to require (at some point in the future) that all public certificates are accompanied by an audit proof. This proof will be a proof that the certificate appears in one or more public logs: if it does not, then this fact will become evident since the proof will not verify. Clients will ship with a list of current logs and a policy for which/how many must be used for each certificate. If a log is shown to have misbehaved, then this list will be updated. This list/policy needs to be widely available so servers know where to publish their certificates.

Because of the audit proof, there is no requirement to trust the maintainer of the public log - if they attempt to fool someone into accepting a certificate that is not logged, then they will have to produce a proof that will not check against the publicly visible version of the log. Assuming we manage to build adequate cross-checks into clients, the discrepancy will become apparent and appropriate action can be taken.

Similarly, there is no need to trust CAs - if they issue a certificate they should not, then one of three things can happen:

1. **The certificate does not appear in a public log and no log colludes with the CA**. In this case, clients will reject the certificate, because it will not be accompanied by an audit proof.
2. **The certificate appears in a public log**. In this case, the client will accept the certificate, but the legitimate owner of the site which the certificate purports to be for will be able to see the mis-issue immediately, and can report the problem and have the certificate revoked.
3. **The certificate does not appear in a public log and at least one log colludes with the CA**. In this case, the colluding log will provide the CA with a fake audit proof. The client may accept this proof temporarily (depending on its ability to communicate with other observers of the public logs) but it should eventually become apparent that the proof is not correct (i.e. does not correspond to the public log). Once this is determined, the audit proof is now a proof that the log did not act correctly, and henceforth should not be trusted.

Clearly in case 3 we will fail, in the short term, to protect users. However, a log should no longer be trusted after it has done this a single time, and so the opportunity for evil logs to compromise users is extremely limited. The opportunity for CAs to do so on their own is completely removed.

Publicly Auditable Certificate Logs

The number of logs is a design decision yet to be taken. There could be one. There could be one per CA. CAs could have nothing to do with logs: they could be provided by independent entities. Or anything in between.

However, each log will operate in the same way, no matter how many they are.

Each log will be a simple append-only list of issued certificates. Each entry will be the end-entity certificate and the intermediate certificates required to validate the end-entity certificate. As the log grows, new entries will be periodically signed using a signed Merkle Tree.[1] The log will be publicly available for download (probably in chunks corresponding to the Merkle signatures). Thus, domain owners can monitor the logs for certificate issuance for their domain. If a certificate is issued that should not have been, then the domain owner will have proof of that fact (the mere existence of the certificate is sufficient). And since browsers will check the Merkle proofs (see below), it will not be possible to issue a certificate that does not appear in the audit log.

[1] That is, every so often, a Merkle tree will be constructed from all new entries since the last time, and the root of that tree signed by the log owner.

Audit Proof

A proof that a certificate is in the audit log is simply the Merkle signature for that certificate - i.e. a list of hashes from the top of the Merkle tree down to the particular certificate, plus a signature on the top hash. In addition to verifying the hashes and the signature, the browser should also verify that the top hash corresponds to the appropriate periodic signature on the public log[2]. Should there be an attempt to forge a public log entry, then one of these checks will fail, and, furthermore, the audit proof will then be a proof of the forgery attempt.

The exact mechanism by which these proofs are presented is to be decided, but two obvious possibilities are:

> An extra certificate, containing the Merkle signature, in the certificate chain presented by the server to the browser. Although this certificate will *not* be part of the validation chain for the server certificate, we believe that most existing verification code will ignore it.
> A TLS extension that shows the Merkle signature during the TLS handshake.

Note that both these mechanisms are backwards compatible with existing browsers, so whilst browsers that do not understand these extensions will gain no direct protection from them[3], they will also not be broken by them. Furthermore, the first mechanism requires only a configuration change in the TLS server, rather than new server code.

Certificate Issuance

Because of the introduction of Merkle signatures, it is necessary to introduce a little latency to certificate issuance. The CA issues a certificate in the usual way, then the site operator (of the site which the certificate is for) sends it to the audit log(s)[4]. When the next periodic signature is made, the site operator can then retrieve the Merkle signature for his certificate from the log. Once that is done, he can configure his server appropriately and use the new certificate.

Back Compatibility

---

[2] Since a log could potentially give different answers to different clients, this hash check needs to be done by as many different routes as possible: e.g. repeatedly over time, via peer-to-peer links with other clients, from third parties and so on. Servers could include a current hash value from their point of view. It might be helpful to have a second Merkle tree made up of the periodic hashes for this purpose.

[3] But note that even unmodified browsers get *indirect* protection by virtue of the early detection of mis-issued certificates.

[4] We anticipate this being done by a simple command-line tool.

As mentioned above, this scheme is completely back compatible with existing browsers: although they will be unable to check the new audit logs and signatures, they will naturally ignore the extra information, and so will be reliant on the existing PKI.

Servers should only require configuration changes, though we would recommend moving to TLS extensions in the longer term, which will require software updates.

Scaling

The major scaling issues are the availability of the logs themselves, and the cost of monitoring those logs. Availability does not seem like a major hurdle, there are many web services with at least as high an uptime requirement as the logs have, and furthermore, because the logs are both signed and public, there is little barrier to services which aggregate and republish logs for improved availability.

Similarly, we anticipate the creation of third-party services for monitoring the logs.

Private CAs

CAs that are added to the trust root by users or administrators can opt out of public audit - this is not a problem from the point of view of protecting the public infrastructure. Similarly it might be possible to allow an intermediate CA to create private certificates within a subdomain - in this case the intermediate CA certificate would have to be logged along with which domain it could create subdomains in, so that mis-issues can still be detected. For example, an X.509 extension specifying the permitted domains could be included in the certificate.

Revocation

We do not address revocation of certificates in this document. However, a similar scheme could be used - an audit log of revoked certificates could be kept, in a similar style to the DNSSEC proof of non-existence of a domain. To create the log, simply list all revoked certificates sorted in some order. Then a proof of non-revocation consists of showing a pair of consecutive entries which bracket the unrevoked certificate. Since this proof would have to be continuously updated, this will require software modification of servers, or extra round-trips in browsers. Because of this we intend to leave revocation for a later phase of the project.

Acknowledgements

Some of this work was inspired by conversations with the EFF about their Sovereign Key project.