Network Working Group
Internet-Draft
Intended status: Informational
Expires: 26 October 2026

B. McMillion

24 April 2026

Server Monitoring of Merkle Tree Certificates
draft-mcmillion-server-monitoring-00

Abstract

This document describes a system for site operators to monitor for mis-issued Merkle Tree Certificates affecting their websites. This monitoring is highly efficient, requiring site operators to do an amount of work that is only logarithmic proportional to the total number of certificates issued. It does this while preserving the security and transparency guarantees of Merkle Tree Certificates.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/Bren2010/draft-server-monitoring.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In the original Certificate Transparency [RFC6962] ecosystem, it was originally intended that site operators would monitor for mis-issued certificates by downloading the entire contents of all trusted transparency logs and filtering out certificates that were irrelevant to them. However, as transparency logs grew dramatically in size, that quickly became prohibitively expensive for both site operators and for the transparency logs themselves.

In place of this, a small number of trusted third-party auditors gained popularity. These auditors would download the entire contents of all trusted transparency logs once, index this data, and allow site operators to query this index at their leisure. This made monitoring more efficient, but introduced a security gap in CT: it made it possible for these third-party auditors, which were not considered a part of the formal WebPKI and were not audited for correct behavior, to misbehave in a way that could prevent site operators from reliably learning about mis-issued certificates.

This document describes a method of monitoring for mis-issued Merkle Tree Certificates that only requires site operators to do an amount of work that scales logarithmically relative to the total number of certificates issued. This ensures that monitoring remains efficient regardless of future growth in the number of CAs and in the number of issued certificates. Importantly, it does this in a way that doesn't introduce new trusted parties, doesn't require excessive coordination between site operators and browsers, and overall maintains the originally intended security model of Merkle Tree Certificates.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

In Merkle Tree Certificates, each Certificate Authority (CA) operates a transparency log that contains all certificates issued by that CA. This transparency log is cosigned by any of a number of trusted cosigners, who are chosen and vetted by the root programs that the CA participates in.

For a site operator that only cares about one (or a small number of) websites, transparency logs are typically considered too expensive to monitor directly. This is because, for a site operator to know that they haven't missed any certificates they might care about, they would need to download every single certificate from every trusted transparency log, which is typically multiple terabytes of data.

A verifiable index is a cryptographic data structure designed to allow verifiable searches, resulting in a "proof of inclusion" or "proof of non-inclusion" for stored data. These proofs allow users to verify that the search results provided by a server are both complete and authentic. This means that a verifiable index operator can download the contents of a transparency log once and integrate all of the certificates that it contains into their index. This index can then be efficiently queried by a large number of site operators, without the verifiable index operator needing to be trusted by the site operators to provide correct results.

Assuming that the index was constructed correctly, site operators that search for their domain in the verifiable index and successfully validate a proof of inclusion are guaranteed to have received all of the certificates that exist for their domain, even if they only

downloaded a small amount of data. Verifying that an index was constructed correctly can be done by any third-party auditor by downloading the same contents of the CA transparency log, recomputing what the root hash of the verifiable index should be, and checking that this matches what the verifiable index operator has published.

3.1. Trusting a Verifiable Index

To be able to query a verifiable index and trust the results, site operators need some reliable way to learn the "correct" root hash of the verifiable index. To avoid the security risk and operational overhead of introducing a distinct set of cosigners specifically for verifiable indexes, this document bootstraps trust from the existing cosigners that the CA already uses.

The way this works is that, at the CA's discretion, they add an accumulated log entry to their transparency log. This log entry contains the root hashes and contact information of several verifiable indexes, as submitted to the CA by the operator of each verifiable index. Since these root hashes are stored in log entries in the CA's transparency log, they end up cosigned by the same cosigners that browsers rely on to authenticate certificate issuance.

Finding accumulated log entries. For site operators to be able to rely on the root hashes stored in these accumulated log entries, they need to be able to find them efficiently without relying on a trusted party. To solve this, all log entries in a CA's transparency log contain a counter. This counter stores the number of accumulated log entries that have been sequenced in the log so far, meaning that it increases by one each time a new accumulated log entry is added, and all subsequent log entries have that same counter until the next accumulated log entry is added.

This allows a site operator, knowing only the root hash of a CA's transparency log, to do a binary search for the most recent accumulated log entry that's been added to the log. Doing a binary search through all log entries is highly efficient and ensures, since binary search is deterministic, that all users see the same most recent accumulated log entry. This prevents a CA from giving different results to different users, such as showing one user incorrect or outdated verifiable index root hashes. It also prevents a CA from claiming that no such log entries exist, to try to trick a user into falling back to a less secure system.

Authenticating transparency log root hashes. As mentioned above, finding the correct accumulated log entry is possible assuming that site operators have an authentic root hash for a transparency log. Ideally, site operators would be able to receive and verify

cosignatures on a root hash in the same way (meaning, with the same cosigner public keys and with the same policy for what constitutes an acceptable set of cosigners) as a browser. However, it's usually prohibitively difficult for a site operator to use a browser's code directly, or to follow a browser's development closely enough to reliably duplicate their certificate verification logic. Instead, this document relies on the fact that when a site operator serves a Merkle Tree Certificate, if browsers successfully connect and accept that certificate, then it must be the case that the root hash in that certificate is cosigned correctly according to the browser's policy. This is an indirect, but very simple and low maintenance way for the site operator to know that a root hash is trustworthy.

4. Protocol

4.1. Accumulated Log Entry

This section defines the format of the accumulated log entries.

Define a codepoint for the accumulated log entry type.

Describe a log entry format that contains a list of the following, for each verifiable index:

* URL

* Root hash

* A list of the following, for each CA monitored by the index:

  - CA trust anchor id

  - The range of indexes in the CA's transparency log that have been indexed. (This is used by auditors when verifying the verifiable index root hash.)

4.2. Certificate Authority Interaction

This section defines the request-response protocol where a site operator requests the most recent accumulated log entry from a CA or a mirror.

Request parameters:

1. The greatest tree size that the requester has previously received from this endpoint, if any.

2. Optional: an additional tree size to prove consistency with. This is used to request an additional consistency proof with the tree head in a site operator's certificate.

Response parameters:

1. The most recent tree head, signed by the CA.

2. A consistency proof between request parameter 1 and the most recent tree head.

3. If request parameter 2 was provided: a consistency proof between request parameter 2 and the most recent tree head.

4. The log entries corresponding to the algorithm in Section 4 of [KEYTRANS].

5. The log entries, other than those already provided in response parameter 4, corresponding to a binary search for the first log entry whose counter equals the greatest value observed in response parameter 4.

6. An inclusion proof for all log entries provided in response parameters 4 and 5.

Requester verifies that:

1. The expected number of log entries was provided and all have monotonic counters.

2. The terminal log entry of the binary search in response parameter 5 has accumulated type.

3. The consistency and inclusion proofs in response parameters 2, 3, and 6 evaluate as expected.

Requester retains the presented tree head and the greatest log entry counter observed.

4.3. Verifiable Index Interaction

This section defines the request-response protocol where a site operator searches for certificates related to its domain in a verifiable index.

Request parameters:

1. Domain to search for.

Response parameters:

1. List of certificates

2. Inclusion proof

Requester verifies that:

1. Evaluating the inclusion proof produces the expected root hash.

5. Security Considerations

At a high level, each step of the monitoring process is secured as follows: Site operators know that a CA transparency log root hash is authentic because they can observe it being accepted by browsers. With an authentic transparency log root hash, site operators are able to deterministically (through binary search) find the most recent accumulated log entry. In the most recent accumulated log entry, site operators are able to learn the root hashes and contact information for various verifiable indexes. With the root hash of a verifiable index, site operators are able to query the verifiable index and verify that the result they get back is a complete and authentic list of all certificates for their domain.

Any third-party auditor is able to crawl a transparency log and confirm that each verifiable index is constructed correctly, that the accumulated log entries contain the expected root hash for each verifiable index, and that the counter in each log entry containing the number of sequenced accumulated log entries is always correct.

If a site operator attempts a binary search for the most recent accumulated log entry and finds that none exist, or that no new accumulated log entry has been created since the site operator’s last query, the binary search process authenticates that result. That is to say, it’s not possible for a CA to claim that an accumulated log entry does not exist to avoid providing it, potentially triggering the site operator to fallback to a less secure system.

6. Comparison to Alternatives

Instead of having a log entry counter, why not have site operators scan the tail of the transparency log looking for the first accumulated log entry? This is an amount of work that would scale linearly with the number of issued certificates and could easily become prohibitively expensive for site operators. Having a trusted party directly tell the site operator the index of the most recent accumulated log entry solves the efficiency problem but reintroduces the security gap discussed in the introduction. Specifically, this

trusted party could incorrectly claim that no accumulated log entry exists, or that no new accumulated log entries have been created since the site operator's last request, to interfere with a site operator's monitoring.

Instead of having a log entry counter, why not enforce that every X log entries is an accumulated log entry? The primary issue with this approach is communicating the parameter X to servers. This parameter would very likely vary by CA. However, unlike browsers, servers don't necessarily trust the CA and likely wouldn't have a trustworthy channel capable of pushing down such information. In practice, the parameter X would need to be communicated to the server through ACME and would be otherwise unauthenticated. This allows a CA to trivially downgrade server monitoring by claiming X is unset or unrealistically large.

A second point worth mentioning is that this approach requires committing, at the time when a CA is first created, to X. If this parameter is too small, the CA will need to sequence an excessive number of these log entries. If the parameter is too large, the CA would either need to accept large delays in monitoring or frequently sequence a large number of null log entries, to get to the next Xth log entry, such that it can sequence an accumulated log entry.

Having a log entry counter prevents trivial downgrade attacks and adjusts smoothly to changes in issuance frequency.

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[KEYTRANS] McMillion, B. and F. Linker, "Key Transparency Protocol", Work in Progress, Internet-Draft, draft-ietf-keytrans-protocol-04, 16 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-keytrans-protocol-04>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

8.2. Informative References

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/rfc/rfc6962>.

Acknowledgments

TODO acknowledge.

Author’s Address

Brendan McMillion
Email: brendanmcmillion@gmail.com