Public Notary Transparency Working Group
Internet-Draft
Intended status: Experimental
Expires: April 22, 2016

B. Ford
EPFL

October 20, 2015

Collectively Witnessing Log Servers in CT
draft-ford-trans-witness-00

Abstract

This document proposes a backward-compatible extension to CT enabling log servers to obtain compact collective signatures from any number of well-known "witness" servers, which clients can check without gossip to verify that log server records have been widely witnessed. Collective signatures proactively protect clients from man-in-the-middle attackers who may have stolen the private keys of one or more log servers, even if the attacker controls the client's network access, the client is unwilling to gossip for privacy reasons, or the client does not wish to incur the network bandwidth and/or latency costs of gossip.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction and Rationale

Certificate Transparency's main security benefit fundamentally relies on public logging of certificates, allowing certificate owners and clients to cross-check and detect certificate misuse. The log servers responsible for this public logging unfortunately represent a new potential class of Single Point of Failure (SPOF), whose private keys may become a new potentially attractive hacking target. For example, if a hacker or powerful adversary were to obtain both a CA's private key and a log server's private key, then the combination of those two keys can potentially be used in Man-In-The-Middle (MITM) attacks against unwitting clients by creating not only falsified certificates but falsified logs (including fake SCTs and STHs) solely for the consumption of the victim.

1.1. The Challenge of Keeping Logs Honest

While CT includes a gossip protocol to help "keep logs honest" and enable nodes to cross-check their worldviews, gossip can protect only well-connected hosts that are able to, willing to, and can devote the time to communicate regularly with multiple independent monitor and auditor servers on the Internet in order to cross-check the structure and consistency of observed logs. This well-connectedness assumption can fail to hold - or fail to be useful - in a variety of scenarios:

- If the client is located in a repressive country in which essentially all available network access is controlled by a government-imposed firewall that persistently MITM-attacks one or more clients and blocks access to independent auditors and monitors outside the country, then the attacker can separate the victim clients from the well-connected Internet and prevent detection for a potentially extended period of time (e.g., until one of the targeted clients leaves the country).

- If the attacked client is a non-mobile device (e.g., a desktop PC) always connected via the same attacker-compromised network access path, then an attacker can similarly keep the victim persistently oblivious to the difference between its CT worldview and the well-connected world's.

- Even when feasible, unrestricted gossip can compromised privacy, forcing on clients an unfortunate choice between greater security and worse privacy (by using one or more trusted auditors that effectively learn the client's browsing behavior) or greater privacy and worse security (by declining the use of a trusted auditor and hence being unable to cross-check SCTs that may have been signed by compromised log-server keys).

- Even when feasible, gossip takes time and consumes network bandwidth, making it impractical for most applications (e.g., web browsers) to delay the acceptance and use of a certificate until gossip-based cross-checking of the certificate has been performed. This inherently leaves a window of vulnerability between exploit and detection, which a savvy attacker can use to obtain the "keys to the kingdom" within even a short window (e.g., a critical password communicated via SSL).

- It has been proposed to use CT to help increase the security of software distributions as it does for certificates - but if an attacker can use a stolen pair of CA and log-server keys "even once" to convince a victim to accept a falsified software update, then that software update can simply disable CT or more subtly modify its configuration to ensure that future gossip by the victim will not notice anything amiss or raise an alarm.

- If the client is a stateless mobile device - such as a laptop running the Tor-based Tails software distribution used for anonymous communication by journalists, whistleblowers, and dissidents - then the mobile device might be MITM-attacked while the victim is at a compromised Wifi cafe, and fail to detect any inconsistency in CT's worldview when it is next booted at a different network access point due to the (deliberate) loss of state.

One existing way to raise the bar to the attacker is to require CT certificates to contain SCTs from multiple independent, well-known log servers. Indeed, Google Chrome already requires three SCTs for EV certificates. However, it is not clear that hacking or otherwise obtaining even three log server keys is necessarily out of the reach of some powerful but realistic attackers. Furthermore, if any version of any CT-enabled client accepts (perhaps non-EV) certificates with a single SCT, then a MITM attacker holding even a single log-server key can form a "downgrade attack", impersonating a site whose proper certificates normally have multiple SCTs but presenting the victim with a fake (non-EV) certificate with only one SCT.

1.2. Proactive Witnessing of Logs

To strengthen CT and address scenarios such as those above, we would prefer that clients (as potential attack victims) be able to check proactively, rather than only retroactively via gossip, whether an SCT or the log tree it resides in has been "widely witnessed" in public, e.g., by the well-connected cloud of audit servers that CT already assumes will exist to check each log server for misbehavior or equivocation. This would ensure that even a MITM attacker holding a CA key and one or a few log-server keys could not make a client accept a fake log without also compromising a (likely significantly larger) number of each log's cloud of auditors as well.

As a first straw-man solution, we might demand that log servers not only sign SCTs themselves but, while generating an SCT, communicate with a threshold number of servers among some well-known group of "co-signing auditors" which we will call "witnesses", and include those witnesses' signatures in the SCT along with the log-server's own signature. This would multiply the size of each SCT by a potentially substantial factor, however, and similarly multiply the computational cost on clients to check these signatures (which may result in a non-trivial power cost on mobile devices). Furthermore, the log-server would need to delay the signing of each SCT to allow for active, online communication with its witnesses, which may add unacceptable delays to SCT creation and may create scalability and performance challenges if the log-server needs to create and log new SCTs at a high rate.

A second straw-man solution addresses the last problem above by expecting log servers to obtain a number of co-signatures from witnesses only on STHs, rather than on individual SCTs. This keeps SCT creation quick and lightweight, imposing online communication costs on only the relatively infrequent and delay-insensitive STH generation operation, which needs to be done only once every few minutes to log an arbitrarily large batch of new SCTs. Obtaining co-

signatures on STHs in this way will protect clients from the types of MITM attacks discussed above provided a mechanism is also added to CT by which clients can request from web sites and check inclusion proofs to verify the relationship between a (singly-signed) SCT and a (multiply co-signed) STH. However, while this is a step forward, it still multiplies the number of expensive signature-checks clients must perform when receiving such an STH from a server.

1.3. Efficient Proactive Witnessing with Collective Signatures

To make proactive witnessing practical and efficient at larger numbers of witnesses - and hence higher security levels - we would like to "compress" all of an STH's (potentially many) witness signatures into one. Multisignatures, theoretically well-understood variations of standard signing schemes, already provide this capability in principle [MULTISIG]. These schemes do not generally scale beyond small signing groups, providing a limited advantage over simply attaching multiple separate signatures as discussed above.

However, methods are now available to scale multisignature generation efficiently to hundreds or thousands of participants, through the use of communication trees and other optimizations [COSI]. In this approach, a log server coordinates with a potentially large number of participating witness servers to form and attach a single collective witness signature to each STH. Clients verifying the STH (or an SCT with an inclusion proof rooted in the STH) need normally perform only two expensive public-key operations: one to check the STH's conventional individual log-server signature, the other to check the collective signature of the witnesses. The log-server's individual signature could in principle be rolled into the collective signature as well, but keeping them separate simplifies backward compatibility.

2. STH Collective Signing Extension

To support collective signing of STHs, we specify a new SthExtensionType (value TBD), whose content is a collective signature generated by one round of the CoSi colllective signing protocol [COSI] initiated by the log-server but run with the cooperation of the log-server's well-known group of public witnesses.

CT's current mechanism for STH extensions presents a minor challenge in that all extensions are defined as being covered by the log server's conventional digital signature (see the definition of SignedTreeHead). This implies that to include a collective witness signature as an SthExtension, the log-server must form the collective witness signature before computing its own individual signature over the full STH content including the witness signature. This in turn implies that the log-server must invoke the CoSi protocol to sign a

slightly different version of the SignedTreeHead content, with the collective witness signature extension omitted (necessarily since it hasn't been computed yet).

A potentially cleaner way to address this issue would be to divide the SthExtensionType namespace into designated ranges denoting "signed" versus "unsigned" extensions, the latter being explicitly excluded from the message on which either individual signatures or collective signatures are computed. This would allow the STH's individual and collective signature to be computed more consistently on the "same" SignedTreeHead content.

2.1. Availability and Signing Thresholds

A natural operational risk is that a log-server might at a given time find that one or more of its well-known witness servers is offline. The CoSi protocol incorporates availability protection mechanisms ensuring that the initiator (the log server in this case) can produce a valid collective signature regardless of which or how many witness nodes are only, but the produced signature will contain metadata documenting which witness nodes were offline at STH-signing time and enabling clients to verify the signature without those witnesses' signature contributions.

A benefit of this availability protection mechanism is that the log server can protect its own progress from unreliability and even DoS attacks on or by witnesses, in principle even if many, most, or all witnesses go offline. It is then ultimately up to client security policy to determine how many witnesses may have been offline (or must have been online) during signing in order for the client to trust the STH.

A cost of this availability protection mechanism, however, is that the size and verification cost of the collective signature is proportaional to the number of witnesses that were missing at signing time. For this reason, log-server operators are expected to choose reliable witness servers run by competent, respected operators who can be expected to keep their witness servers online consistently. Provided almost all witness servers are online at any given time, the produced STH collective signature is barely larger than a single individual signature.

2.2. Identity of a Log Server's Witness Group

A log-server's group of witnesses cannot be a "wide-open" group, since an attacker who can add any number of bad witnesses to the group could perform a Sybil attack by adding a threshold number of malicious witnesses that collude to produce collective signatures

that clients will accept. Thus, the operational expectation is that log-servers specify a public, relatively stable, reputable, and transparent set of witness servers for the log server to use.

In order for clients to check the log's collective witness signatures, the clients must of course "know" the group of witnesses with which the log server collectively signs its STHs. For this purpose, clients that support collectively-signed STHs must include in their roots of trust, alongside the log-server's public key, a collective public key representing the aggregate of all the log-server's witnesses. Like collective signatures, this collective public is small and independent of the number of witnesses, amounting to a single elliptic-curve point and a single cryptographic hash. (The hash represents the root of a Merkle tree containing all witness servers' individual public keys plus additional data needed in the availability protection mechanism [COSI]).

2.3. Evolution of Witness Groups

A log server's set of witnesses must also of course change occasionally, perhaps once per year in the long-term, or somewhat more frequently during initial development and testing. Just as conventional CA and log-server keypairs are typically valid for overlapping multi-year windows, a log-server's collective public key may be refreshed and gradually rolled over in similar fashion, via the usual process of updating the relevant client software (e.g., web browser) containing the log server in its root of trust.

Collective signing presents a potentially more attractive alternative, however. When it comes time to evolve a log server's witness group, the log server operator first produces and announces the public key for the new witness group. This new collective witness key can and perhaps should be based on new individual public keys freshly generated by the individual witness servers. Then, as the final collective signature produced in the old group, the log server initiates the collective signing of a collective "forward-pointer" attesting that the new collective public key is the one and only valid successor to the old group's public key. Finally, once this collectively signed forward-pointer is announced, all witness nodes in the new and old group securely erase the private keys representing their portions of the old collective public key.

Through these collectively signed forward-pointers, clients with old software (containing old roots of trust) can "chain forward" from the last collective witness group they know to the latest one by retrieving and following a few such links. Provided witness groups do not change too often (e.g., once a year), clients will not need not follow too many such forward-pointers unless they are so out-of-

date that the security of their software and crypto is likely suspect anyway.

3. Security Considerations

This draft contains nothing but security considerations.

4. References

4.1. Normative References

[COSI] Syta, E., Tamas, I., Visher, D., Wolinsky, D., and B. Ford, "Decentralizing Authorities into Scalable Strongest-Link Cothorities", March 2015, <http://arxiv.org/abs/1503.08768>.

4.2. Informative References

[MULTISIG] Micali, S., Ohta, K., and L. Reyzin, "Accountable-Subgroup Multisignatures", ACM Conference on Computer and Communications Security 2001, August 2001, <http://cs-www.bu.edu/~reyzin/papers/multisig.pdf>.

Author's Address

Bryan Ford
EPFL
BC 210, Station 14
Lausanne CH-1015
Switzerland

Phone: +41 21 693 28 73
Email: bryan.ford@epfl.ch