No Root Store Left Behind

James Larisch
Harvard University

Waqar Aqeel
Duke University

Taejoong Chung
Virginia Tech

Eddie Kohler
Harvard University

Dave Levin
University of Maryland

Bruce M. Maggs
Duke University &
Emerald Innovations

Bryan Parno
Carnegie Mellon University

Christo Wilson
Northeastern University

Abstract

When a root certificate authority (CA) in the Web PKI misbehaves, primary root-store operators such as Mozilla and Google respond by distrusting that CA. However, full distrust is often too broad, so root stores often implement *partial distrust* of roots, such as only accepting a root for a subset of domains. Unfortunately, derivative root stores (e.g., Debian and Android) that mirror decisions made by primary root stores are often out-of-date and cannot implement partial distrust, leaving TLS applications vulnerable.

We propose augmenting root stores with per-certificate programs called General Certificate Constraints (GCCs) that precisely control the trust of root certificates. We propose that primary root-store operators write GCCs and distribute them, along with routine root certificate additions and removals, to all root stores in the Web PKI. To justify our arguments, we review specific instances of CA certificate mis-issuance over the last decade that resulted in partial distrust of roots that derivative root stores were unable to precisely mirror. We also review prior work that illustrates the alarming lag between primary and derivative root stores. We discuss preliminary designs for GCC deployment and how GCCs could enable pre-emptive restrictions on CA power.

CCS Concepts

• **Security and privacy** → **Web protocol security**; Logic and verification; • **Networks** → Transport protocols.

Keywords

SSL/TLS, Web PKI, X.509 certificate, root store

ACM Reference Format:
James Larisch, Waqar Aqeel, Taejoong Chung, Eddie Kohler, Dave Levin, Bruce M. Maggs, Bryan Parno, and Christo Wilson. 2023. No Root Store Left Behind. In *The 22nd ACM Workshop on Hot Topics in Networks (HotNets '23), November 28–29, 2023, Cambridge, MA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626111.3630268

HotNets '23, November 28–29, 2023, Cambridge, MA, USA

ACM ISBN 979-8-4007-0415-4/23/11…$15.00
https://doi.org/10.1145/3626111.3630268

1 Introduction

Transport Layer Security (TLS) *root certificate stores* anchor the security of the Web. Before finalizing a TLS connection to a given server, user-agents (e.g., browsers and TLS libraries) validate the server's X.509 certificate *chain*. Each chain is a collection of certificates, starting from the server's end-entity or *leaf* certificate and ending with a trusted *root* certificate authority (CA) certificate, with zero or more subordinate or *intermediate* CA certificates in between. Root certificates[1] are distributed as collections called root stores.[2] User-agents use either their own root store (e.g., Firefox) or the root store managed by the underlying operating system (e.g., Debian or Android) when validating certificate chains.

Despite the critical importance of root certificate stores for web security and privacy, most root stores suffer from critical management and technical challenges. We refer to these problems as *lag* and *imprecision*.

Lag. Most non-browser root stores are out-of-date. Because *primary* root stores like Mozilla's are well-managed and up-to-date, many platforms (e.g., Debian, Android, and Amazon Linux) manually mimic Mozilla's root inclusion and exclusion decisions. Unfortunately, Ma et al. showed that, as of 2021, most of these *derivative* root stores are often months behind their upstream primaries [44], leading to vulnerability windows when the primary store removes a root certificate in response to a CA compromise. domains (e.g., banks or email services).

Imprecision. The second and more critical problem is that primary root stores are no longer mere collections of certificates. A collection of certificates can only express a binary trust decision: either the root is in the store (and thus trusted), or it is not. In reality, primary root stores are collections of certificates augmented with granular, certificate-specific policies. For instance, in 2018, Firefox implemented *partial* distrust of problematic Symantec roots by hard-coding trust

[1] Technically certificate chains must terminate in a trust anchor, which are often distributed as X.509 certificates.
[2] We ignore 'imported' roots added by users or private networks, though prior work has investigated their use and misuse [54].

or distrust of particular descendant certificates issued at certain times and by certain subordinate CAs [13]. Certificate revocation in the Web Public Key Infrastructure (PKI) [43], despite recent improvements [39], also remains an all-or-nothing decision.

Unfortunately, derivative root stores are all-or-nothing and cannot mimic non-binary decisions made by primary root stores. A root certificate in a derivative root store is either fully trusted or not. This disparity can cause security problems and operator confusion. For instance, when Firefox implemented partial distrust of Symantec roots, Debian had no choice but to completely remove the affected roots—a decision they were later forced to revert after user complaints [15, 44].

Compared with derivative root stores, primary root stores like Mozilla and Google have a distinct advantage: their operators also distribute popular web browsers that can layer the partial distrust of roots on top of their simple collections of root certificates. Debian, in contrast, can only expose a collection of certificates (`/etc/ssl/certs`) to applications, who are in turn responsible for performing full chain validation. Applications (e.g., `curl`) or TLS libraries (e.g., OpenSSL) built atop the platform must either manually replicate the partial distrust implemented by primaries such as Firefox or remain subject to the underlying derivative platform's limited, binary decisions.

Our proposal. We propose a framework for both describing precise, root-specific trust policies and for distributing these policies to all root stores, regardless of platform, in a timely manner. Primary root stores are already augmented with small de facto programs that constrain particular root certificates. We propose that primary root stores encapsulate these constraints into portable, per-certificate programs called *General Certificate Constraints* (GCCs). We also propose a standardized mechanism for primary root stores to quickly push changes (both routine certificate additions/removals and GCCs) to derivatives. These mechanisms will help minimize the vulnerability window after a root CA is deprecated, the incompatibility window when a new root CA becomes trusted [30], and the discrepancy between the all-or-nothing trust of derivatives and the more nuanced trust of primary root stores.

Deploying new mechanisms for precise, up-to-date root-store distribution presents significant challenges. The biggest challenge is that GCCs must be executed during certificate validation, which may require changes to the interface between TLS applications and the platform's underlying root store. Another challenge is that subscribe-able root-store "feeds" require standardization, and must also be considered critical security infrastructure.

2 Background & Motivation

Before proposing new mechanisms for deploying certificate distrust to all root stores, we review the problems with how root stores are managed in the Web PKI today.

2.1 Root Stores in the Web PKI

The Web PKI has a fundamental weakness: CA certificates (both intermediate and root) are allowed to sign virtually *any* certificate. A compromised or rogue root CA can launch monster-in-the-middle (MITM) attacks by masquerading as the target website using a fraudulent certificate valid for the website's domain. Certificates have been fraudulently issued in the wild—in 2011, a Comodo CA partner was compromised and nine fraudulent certificates were issued for domains including `google.com` and `addons.mozilla.com` [5, 20].

Root-store operators like Mozilla monitor CA issuance for fraudulant or negligent behavior. When a *root* CA misissues certificates, Mozilla may respond by removing the root certificate from their store altogether. For instance, in 2016 Mozilla performed an in-depth investigation into compliance violations of WoSign, a root CA, eventually leading to the removal of WoSign's root certificate [12]. Historically, Mozilla has been the most responsive root-store operator [44], with stringent requirements on CA behavior [48]. Mozilla's root store is stored in their Network Security Services (NSS) [7] library, which is included in Mozilla Firefox.

However, most root stores do not monitor CA mis-issuance. Instead, they mimic decisions made by historically responsive root-store operators like Mozilla. In 2021, Ma et al. found that many systems (e.g., Debian/Ubuntu, Android, Alpine Linux, Amazon Linux, and NodeJS) base their root stores off NSS due to Mozilla's comprehensive due diligence [44]. These platforms periodically (and manually) mirror decisions made by NSS (removals and additions) via software updates. Ma et al. classify such root stores as *derivative* root stores and classify upstream sources like Mozilla's NSS as *primary* root stores. We reuse their terminology.

2.2 Root Trust is Not All-or-Nothing

However, primary root stores sometimes deploy *partial* distrust of certificates to limit the Web's exposure if a root is compromised, or to avoid the collateral impact of complete removal after a root is compromised. Mozilla uses two mechanisms for systematic partial distrust: date-usage constraints and EV constraints. First, NSS attaches a date-usage pair to each root that signifies the last date a leaf certificate transitively signed by that root can be used for the given usage (either TLS or S/MIME). Second, Firefox records whether each root can issue Extended Validation (EV) certificates.[3]

However, Mozilla often implements additional, ad hoc partial distrust of root certificates by hard-coding certificate-specific logic directly into Firefox or the NSS library that Firefox includes. Below we list root certificate incidents from the last decade, as well as the resulting ad hoc distrust implemented by Mozilla (or by Google, which did not manage their own root store at the time, but still hard-coded restrictions for known roots).

[3]EV certificates were certificates for which the issuing CA performed additional validation checks on domain ownership, and are currently being phased out by Chrome and Firefox [27].

TurkTrust. In January of 2013, Google discovered that the Turkish government root CA TURKTRUST misissued two intermediate CA certificates, one of which issued a leaf certificate for `*.google.com` [36]. In response, Google revoked the intermediates via CRLSet [8] and updated Chrome to disallow EV certificates from TURKTRUST. Mozilla later followed suit [47]. Later (and perhaps unrelatedly), when another Turkish government root CA (TUBITAK) applied for root inclusion, Mozilla added a hard-coded name constraint to NSS that allows the new root to issue leaf certificates for Turkish government top-level domains (TLDs) only [11].

ANSSI. In December of 2013, Google discovered that the root certificate of French government agency ANSSI misissued an intermediate CA certificate that issued at least one leaf certificate for Google domains [28]. In response, Google and Mozilla revoked the intermediate and later name-constrained the ANSSI root to French government TLDs [6, 37].

India CCA. In July of 2014, Google detected misissued leaf certificates for several Google domains and a Yahoo domain issued by intermediate CA certificates signed by the India Controller of Certifying Authorities (CCA) [17]. The India CCA certificate was included in the Microsoft root store (and thus Firefox was unaffected). In response, Google revoked the misissued leaves via CRLSet and later revoked the offending intermediates. Chrome later name-constrained the India CCA root to Indian TLDs.

MCS/CNNIC. In 2015, Mozilla and Google were notified that an intermediate signed by the China Network Information Center (CNNIC) was being used to MITM traffic. In response, CNNIC revoked the offending intermediate, which was held by MCS Holdings. Mozilla and Google revoked the MCS intermediate via OneCRL and CRLSet, respectively [9, 18]. Later, they partially distrusted the CNNIC root with an allowlist of exempted subordinate certificates [9].

WoSign/Smartcom. In October 2016, Mozilla discovered that the WoSign CA backdated 62 SHA-1 certificates to circumvent the new CA/B Baseline Requirements that forbid the use of SHA-1 after 2015 [12]. They also discovered that WoSign had covertly acquired the CA Smartcom and failed to disclose the acquisition, in violation of Mozilla's root-store policy.[4] In response, Mozilla distrusted all *new* leaf certificates chaining up to the offending roots [10] (maintaining existing leaves). They also revoked all backdated leaf certificates via OneCRL. Google followed suit [19].

Symantec. In early 2018, Google [23] and Mozilla [14] (among others) implemented a plan for the gradual distrust of one of the oldest and largest CAs at the time, Symantec (and subsidiary CAs GeoTrust, RapidSSL, and Thawte), due to a number of compliance issues [2]. As of May 2018, Firefox distrusted leaf certificates issued after June 1st, 2016 that chained up to the affected roots. As of October 2018, Firefox distrusted all leaf certificates chaining up to the affected roots except for a few allowlisted intermediate CA certificates issued by Symantec roots but controlled by Apple and Google. Chrome followed suit with different dates.

[4] Tracking CA ownership in the Web PKI is a challenge unto itself [45].

2.3 Derivative Stores Lack Partial Distrust

Unfortunately, there is no way for derivative root stores to mimic the above partial distrust decisions because derivative stores are simply collections of certificates—each root is either completely in the store or completely not. If a derivative store includes a root but not the associated hard-coded name or date constraints, dependent clients may accept fraudulent or mis-issued chains that the primary would not. On the other hand, if a derivative store completely excludes a partially distrusted root, clients may experience denial of service. In 2018, Debian imprecisely mimicked Mozilla's partial distrust of Symantec roots by simply removing them from their store, resulting in collateral service disruption that forced them to completely restore the roots [15, 44].

3 General Certificate Constraints

To bring precise root trust to the entire Web PKI, we propose that primary root-store operators translate their root-specific constraints into standardized programs called General Certificate Constraints. A GCC is a simple program attached to a specific root certificate (by SHA-256 hash) that returns a Boolean true or false. If the GCC returns false, the certificate chain in question must be rejected. Each GCC should be distributed by primary root stores to derivative root stores and executed during the construction of certificate chains that include the associated root.

The PKI could use any programming language to express GCCs, but we find that logic programming languages, specifically stratified Datalog [16, 52], fit well for many reasons. First, Datalog has been used extensively for both expressing [22, 26, 33, 35, 49, 53] and reasoning about [21, 25, 40–42] authorization languages. Second, Datalog is declarative and based on first-order logic, making its semantics easy to reason about. Finally, Datalog termination is guaranteed, and it lacks I/O capabilities—these properties help minimize the threat of evaluating arbitrary code.

GCC execution happens during certificate chain construction and validation: a constructed chain is valid if and only if all GCCs attached to the candidate root are valid. A GCC operates over the fields of all certificates in the candidate chain. To execute a GCC, the chain in question is first converted into a form the GCC program can read. For stratified Datalog, this means converting each X.509 certificate field into a Datalog statement. Further, relationships between certificates (i.e., that a particular certificate signs another) must also be codified as Datalog statements. Next, the converted statements are fed, along with the GCC in question, into the Datalog interpreter. To execute, the validator performs the following Datalog query: `valid(Chain, Usage)?`, which checks whether the required GCC `valid` rule holds for the

```
nov30th2022(1669784400). % Unix timestamp
valid(Chain, "S/MIME") :- % Valid rule for S/MIME usage
  leaf(Chain, Cert), % Get the chain's leaf certificate
  nov30th2022(T), % Get November 30th, 2022
  notBefore(Cert, NB), % Get the leaf's notBefore date
  NB < T. % Holds if notBefore before November 30th, 2022
valid(Chain, "TLS") :- % Valid rule for TLS usage
  leaf(Chain, Cert), % Get the chain's leaf certificate
  \+EV(Cert), % Assert that leaf is not EV
  nov30th2022(T), % Get November 30th, 2022
  notBefore(Cert, NB), % Get the leaf's notBefore date
  NB < T. % Holds if notBefore before November 30th, 2022
```

Listing 1: Example constraints based on those found on the TrustCor root in NSS. Note that \+ is the 'not' operator in Datalog.

```
june1st2016(1464753600). % Unix timestamp
exempt(...).
valid(Chain, _) :-
  leaf(Chain, Cert), % Get the chain's leaf
  notBefore(Cert, NB), % Get the leaf's notBefore date
  june1st2016(T), % Get June 1st, 2016 date
  NB < T. % Holds if notBefore date is before June 1st, 2016
valid(Chain, _) :-
  root(Chain, Root), % Get the chain's root
  signs(Root, Int), % Get the intermediate signed by root
  hash(Int, H), % Get the intermediate's SHA-256 hash
  exempt(H). % Holds if hash is one of exempt hashes
```

Listing 2: NSS constraints on Symantec roots as of May 2018 expressed as a GCC. A chain terminating in a Symantec root is valid if the leaf is issued before 2016 or the first intermediate is exempt.

given converted chain of certificates under the given usage ("TLS" or "S/MIME").

All of the systematic constraints that Mozilla places on root certificates can be expressed using GCCs. Listing 1 shows an example GCC that mimics the constraints on a TrustCor root [46, 50] currently found in NSS. Once attached to the TrustCor root certificate, this GCC signifies that for "S/MIME" usage, the certificate chain is only valid if the leaf certificate was issued before November 30th, 2022. For "TLS", the constraint is the same except the certificate must also not be EV. Mozilla could write a similar GCC for every root in NSS that has a date/usage constraint.

GCCs can also be used to express the more ad hoc root constraints implemented in response to incidents. For instance, Listing 2 shows NSS's constraints on Symantec roots as of May 2018. A chain including a Symantec root is valid if either **(1)** the leaf was issued before June 1st, 2016, or **(2)** if the first intermediate (signed by the root) is one of the exempt intermediates (identified by SHA-256 hash).

3.1 Chain-Validation Must Change

Unfortunately, GCCs require changes to TLS certificate-chain construction and validation algorithms: whenever a candidate root is found with a GCC, the validator must execute the GCC to determine whether to accept the chain or continue building. For instance, if OpenSSL on Debian validates a certificate chain that chains up to a Symantec root, it must validate the root's GCC to check for the specific circumstances under which the Symantec root should be accepted.

While GCCs can be deployed on a per-certificate basis, the critical question of who should execute GCCs (user-agent or platform) remains open. We consider three options:

User-agent execution. The first option is to push GCC-execution responsibility to TLS user-agents. In this scheme, when user-agents pull root certificates from the underlying platform, they must also pull all associated GCCs (which the platform exposes). During chain validation, the user-agent must convert all certificates into Datalog statements and evaluate any attached GCCs. We performed a preliminary performance analysis in which we measured the time taken to convert ~100K certificates to their respective sets of Datalog statements and found that the mean (unoptimized) conversion time was ~2.4ms. To evaluate GCCs, each user-agent must include a Datalog interpreter to interpret the Datalog certificate statements and execute the query.

While user-agent execution may seem like the most challenging road to widespread GCC deployment, consider that **(1)** incremental client deployment is acceptable because incompatible clients are no worse off than they are today, and **(2)** while there may appear to be many user-agents, we suspect that most use a small set of libraries e.g., OpenSSL. Only these libraries (and the platforms) would need to support GCCs. We leave measuring how common various TLS libraries are across all platforms to future work.

Platform execution. The second option is for platforms to deploy a new system daemon responsible for executing GCCs or augment an existing system daemon like MacOS's `trustd`. This system daemon must expose an IPC interface that accepts certificates and returns a Boolean (valid or not). In this scheme, during chain construction TLS user-agents pass the candidate chain and the selected root to the platform's system daemon. The system daemon converts the chain into Datalog statements and executes all GCCs attached to the selected root, then returns the result to the user-agent, which proceeds with chain construction (building a new chain if the daemon responded `false`).

Platform execution requires major platform changes and minor user-agent changes. The platform must support Datalog execution and expose an IPC interface. The user-agent must be changed to contact the system daemon during TLS certificate validation, though again, if most user-agents use a small set of TLS libraries, this change may be easier.

Complete validation redesign. GCCs may be easier to deploy given a radical redesign of certificate chain validation. For instance, in Hammurabi [38], the entire TLS certificate validation algorithm is expressed as a Prolog program. A Hammurabi-enabled platform could perform the complete chain validation procedure—user-agents would simply pass certificate and the chosen Hammurabi policy (e.g., authored by Mozilla or Google) to the platform's trust daemon, which would perform chain construction and return `true` or `false`.

The trust daemon could easily execute GCCs since it would already include a logic program engine.

4 Distributing Roots & GCCs

GCCs, like routine root additions and removals, must be distributed to derivative root stores in a timely, secure fashion. Unfortunately, most derivative root stores today are out of date: Ma et al. found that many derivative root stores are months behind their respective primary stores [44]. They quantified the staleness of derivative root stores by finding the closest primary-derivative match (since derivative root stores often change primary stores slightly) and comparing release dates. They found no derivative root stores that matched NSS's update schedule. Amazon Linux, for instance, "exhibits an average staleness of more than four substantial versions". They also found that "Android is always several months behind". They did find, however, that a few derivatives have responded in a timely fashion to certain major root breaches—though a more in-depth analysis of how quickly, and to which breaches, may still be warranted.

Public root-store feeds. To combat derivative root store staleness, we propose a new mechanism for pushing routine root store changes (certificate additions and removals) as well as GCCs from primary root stores to derivative root stores. We call this mechanism root-store feeds (RSFs): a RSF is a sequence of root-store snapshots where, between snapshots, both certificates and GCCs may be added or removed. Each snapshot may be annotated with justifications of particular decisions and links to public discussions like Mozilla's PKI security message board [3] or Bugzilla [1].

We propose that RSF clients be included as core operating system daemons. For instance, under this new scheme Debian/Ubuntu should write a core RSF systemd service that periodically (hourly) polls the primary RSF of their choice (e.g., NSS) and updates the root certificates exposed to applications as needed. Fortunately, ignoring GCCs, derivative root stores need not change their application interfaces. Debian, for instance, can continue to expose root certificates in `/etc/ssl/certs` and add/remove roots as dictated by Mozilla's RSF updates. Users can still add imported roots—self-signed certificates not part of the Web PKI—by adding these roots to the platform's directory.

RSFs can be distributed using conventional protocols (e.g., HTTPS); however, we highlight three factors that should be taken into account when designing and deploying RSFs.

Negative inclusion. When a root-store operator explicitly *removes* a root, that certificate should have a different status (distrusted) from a certificate that was simply never added (untrusted). We propose that root stores be composed of two sets of certificates: those that are explicitly trusted and those that are explicitly distrusted. This negative root inclusion subsumes root certificate revocation, for which certain browsers have satisfactory solutions [8, 29, 39] but many TLS user-agents lack [43].

RSF merging. Maintaining a distrusted set is important because derivative root stores sometimes augment their primary root stores. For example, Ma et al. found that Amazon Linux re-added 16 root certificates after they had been explicitly removed by NSS [44].

To further acommodate root store augmentation, we propose that derivative root stores publish their own RSFs and that the two RSFs be merged before deployment. This way, if a primary RSF explicitly distrusts a root certificate and a derivative RSF later adds that certificate, the attempted merge flags an issue to the operator: the root in question is in the primary's distrusted set but the derivative's trusted set. We leave more in-depth discussion, including how to prioritize RSFs and whether RSFs should be merged manually or automatically, to future work.

Security. RSFs will be an attractive target for attackers because compromised RSFs can distribute malicious roots that can issue arbitrary leaves. RSFs must thus be treated as critical infrastructure (like browser software update channels). We recommend that RSF updates be signed with a separate key that should itself be signed by a coordinating body like ICANN. We leave the full exploration of RSF security, including the potential use of immutable logs, to future work.

RSFs can be consumed by both derivative root-store platforms *and* browsers. While prior work has shown that derivative root stores are perhaps most in-need of RSFs, a browser may choose to subscribe to its own RSF to ensure freshness. Indeed, browser operators already push some certificate information (revocations) outside of software updates (via CRLSet [8] and OneCRL [29]).

5 Towards Pre-emptive Constraints

So far, we have discussed how, using GCCs and RSFs, primary root stores can push precise and timely updates to the entire Web PKI. In this section, we consider how the security posture of primary operators is largely defensive, and how GCCs may help shift the PKI towards pro-active measures.

5.1 The Lack of Pre-emptive Constraints

Certificate authorities today can, in theory, pre-emptively constrain their certificates to minimize the impact of attacks before they happen. First, CA certificates can include a path-length constraint, which limits the number of certificates allowed in a descending chain—this can be used to prevent a compromised certificate from issuing another intermediate. Second, CA certificates can include name constraints [31], which limit the domains for which descendant leaf certificates may be issued. For instance, a root CA may specify that all descendant leaves, regardless of intermediate, may only be issued for domains matching `*.gov`. If an attacker compromises the root CA, it cannot issue certificates (either directly or via a new, malicious intermediate) for `example.com`. For instance, the Hellenic Academic and Research Institutions [4] root CA has 23 name-constrained subordinate CA certificates, many constrained to Greek domains (`*.gr`).

```
oneMonthInSeconds(2630000).
lifetimeValid(Leaf) :-
  notBefore(Leaf, NB), % Get the leaf's notBefore date
  notAfter(Leaf, NA), % Get the leaf's notAfter date
  Lifetime = NA - NB, % Calculate leaf's lifetime
  oneMonthInSeconds(Limit), % Get one month (in seconds)
  Lifetime <= Limit. % Holds if leaf lifetime is < one month
validUsage(Leaf) :-
  extendedKeyUsage(Leaf, "id-kp-serverAuth"),
  keyUsage(Leaf, "digitalSignature").
valid(Chain, "TLS") :- % Valid TLS usage only
  leaf(Chain, Cert), % Get the chain's leaf certificate
  lifetimeValid(Leaf), % Holds if leaf lifetime is valid
  validUsage(Leaf).
```

Listing 3: Example pre-emptive constraint on a hypothetical root. All leaves under the root will be constrained to TLS `serverAuth` extended key usage, `digitalSignature` key usage, and a one month lifetime.

Unfortunately, CA-driven certificate constraints suffer from a number of issues. First, it is the responsibility of TLS user-agents to honor name constraints specified by CAs, and it took roughly two decades for the name constraints to be widely supported by major user-agents [32]. Indeed, Firefox and OpenSSL have disagreed on the semantics of leading dot characters in name constraints [24].

Second, the majority of CAs today do not name constrain their certificates. We conducted a preliminary measurement of constraints in the PKI today by examining the NSS root store as of July 19th, 2022 and intermediate CA certificates in the Nimbus2022, Argon2022, Argon2023, and Xenon2023 Certificate Transparency [51] logs that were non-expired as of August 2nd, 2022. We found that out of 140 root certificates, zero used name constraints and only five used path-length constraints. Out of 776 intermediate CA certificates, 701 used path-length constraints but only 31 used name constraints. Only six (out of 140) roots were included in at least one chain where an intermediate included a name constraint.

5.2 Pre-emptive GCCs

Rather than waiting for CAs to use the name constraints extension, which is limited to names, browsers could leverage Certificate Transparency and GCCs to constrain all fields of certificates before compromise. We propose browsers and/or root stores pre-emptively construct, for each root, a GCC that limits that root's *scope of issuance*, i.e., the names, lifetimes, key usages, and other fields that it may issue certificates for. This minimizes the damage of a root compromise, since issuing power has already been checked.

Pre-emptive GCCs expand upon the 2013 work, CAge, by Kasten et al. which proposed pre-emptively constraining the names of CA certificates based on their issuance patterns [34]. CAge was built on the observation that most CAs only issue certificates for a small set of top-level domains: 90% of CAs sign certificates for ≤ 10 different TLDs. Using CAge, if a CA issued a certificate for a new TLD for which it has not issued a certificate before, browsers would reject that certificate.

Pre-emptive GCCs have two advantages over CAge. First, CAge limited itself to name constraints, while GCCs allow operators to constrain any (e.g., X.509 field) of certificates chains. Listing 3 shows an example of a pre-emptive GCC. Second, GCCs would be built on RSFs, which can systematically distribute pre-emptive constraints to browsers and other TLS user-agents. We also note that both CAge and GCCs benefit from Certificate Transparency: operators can more easily examine scopes of issuance because all certificates must be publicly logged to be trusted by Chrome.

Determining to what extent PKI stakeholders could constrain CA certificates, both root and intermediate, requires further study. Starting from a given set of roots (i.e., NSS), the study should construct all certificate paths and then determine each CA certificate's scope of issuance. Operators could then construct a GCC for each CA certificate that limits its future issuance to its current scope—e.g., if the CA tries to issue a certificate for a key usage it has never used before, the GCC would cause the certificate to be rejected.

A more in-depth study could discover opportunities for splitting CA certificate responsibility across multiple new, limited certificates. For instance, if a CA exhibits a bi-modal scope of issuance, the CA could potentially be split into two root certificates, each more tightly constrained to its de facto scope. GCCs could encourage CAs to issue more tightly constrained roots, each with a narrower issuing purpose.

6 Concluding Discussion

GCCs and RSFs aim to standardize and distribute precise root certificate controls—which major stakeholders in the Web PKI already exert over roots—to all stakeholders. We believe that such mechanisms should exist because many root stores may be out-of-date and insecure. We hope that this work inspires measurement studies (e.g., examining how insecure derivative root stores are, how many user-agents would require RSF/GCC support, and how tightly CAs could be constrained) and design work (e.g., how to securely distribute RSF updates and how to design GCC execution interfaces).

However, GCCs—especially pre-emptive GCCs—highlight important questions about how power is distributed among stakeholders in the PKI. Nominally, CAs have the power to issue any certificate they wish. In response, browsers like Firefox and Chrome enforce stringent policies that all roots and CAs must follow in order to be accepted by these browsers. They reserve the right to take action if a CA is compromised or violates their policies; should they also have the power to restrict a CA's power before a violation occurs?

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This research was supported in part by a gift from VMware and also NSF grants CNS-1900996, CNS-1901325, CNS-2053363, CNS-2247306, CNS-1901047, and CNS-1900879. We thank Zachary Newman for his helpful feedback on early drafts.

References

[1] [n. d.]. Bugzilla. ([n. d.]). https://bugzilla.mozilla.org/home.

[2] [n. d.]. CA/Symantec Issues. ([n. d.]). https://wiki.mozilla.org/CA/Symantec_Issues.

[3] [n. d.]. dev-security-policy@mozilla.org. ([n. d.]). https://groups.google.com/a/mozilla.org/g/dev-security-policy.

[4] [n. d.]. Harica is a member of the PKI Consortium. ([n. d.]). https://pkic.org/members/harica/.

[5] 2011. Comodo Certificate Issue – Follow Up. (March 2011). https://blog.mozilla.org/security/2011/03/25/comodo-certificate-issue-follow-up/.

[6] 2013. Hard code ANSSI(DCISS) to french gov dns space. (2013). https://bugzilla.mozilla.org/show_bug.cgi?id=952572#c2.

[7] 2014. Network Security Services. Mozilla Developer Network. (2014). http://mzl.la/1DRKqGZ.

[8] 2015. CRLSets. The Chromium Projects. (2015). http://bit.ly/1JPsUeC.

[9] 2015. Revoking Trust in one CNNIC Intermediate Certificate . (March 2015). https://blog.mozilla.org/security/2015/03/23/revoking-trust-in-one-cnnic-intermediate-certificate/.

[10] 2016. Distrusting New WoSign and StartCom Certificates . (October 2016). https://blog.mozilla.org/security/2016/10/24/distrusting-new-wosign-and-startcom-certificates/.

[11] 2016. TUBITAK Kamu Sertifikasyon Merkezi - New Root Certificate. (2016). https://bugzilla.mozilla.org/show_bug.cgi?id=1262809#c33.

[12] 2016. WoSign and StartCom. (2016). https://docs.google.com/document/d/1C6BlmbeQfn4a9zydVi2UvjBGv6szuSB4sMYUcVrR8vQ/edit.

[13] 2017. Mozilla's Plan for Symantec Roots. (October 2017). https://groups.google.com/g/mozilla.dev.security.policy/c/FLHRT79e3XE.

[14] 2018. Distrust of Symantec TLS Certificates . (March 2018). https://blog.mozilla.org/security/2018/03/12/distrust-symantec-tls-certificates/.

[15] 2020. ca-certificates: Removal of GeoTrust Global CA requires investigation. (2020). https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=962596.

[16] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.

[17] Adam Langley. 2014. Maintaining digital certificate security. (July 2014). https://security.googleblog.com/2014/07/maintaining-digital-certificate-security.html.

[18] Adam Langley. 2015. Maintaining digital certificate security . (March 2015). https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html.

[19] Andrew Whalley. 2016. Distrusting WoSign and StartCom Certificates. (October 2016). https://security.googleblog.com/2016/10/distrusting-wosign-and-startcom.html.

[20] Ars Technica. 2011. State-sponsored hackers in China compromise certificate authority. (September 2011). https://arstechnica.com/information-technology/2011/09/comodo-hacker-i-hacked-diginotar-too-other-cas-breached/.

[21] Moritz Becker, Cédric Fournet, and Andrew Gordon. 2007. Design and semantics of a decentralized authorization language. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*. IEEE, 3–15.

[22] John DeTreville. 2002. Binder, a logic-based security language. In *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 105–113.

[23] Devon O'Brien, Ryan Sleevi, Andrew Whalley. 2017. Chrome's Plan to Distrust Symantec Certificates. (September 2017). https://security.googleblog.com/2017/09/chromes-plan-to-distrust-symantec.html.

[24] Differences in openssl and nss interpretations of the leading dot [n. d.]. mozilla/gecko-dev. Github. ([n. d.]). https://github.com/mozilla/gecko-dev/blob/f8087305eb1ebea329b838924627008713c5f56c/security/nss/lib/mozpkix/lib/pkixnames.cpp#L997.

[25] Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2006. Specifying and reasoning about dynamic access-control policies. In *International Joint Conference on Automated Reasoning*. Springer, 632–646.

[26] Eslam Elnikety, Aastha Mehta, Anjo Vahldiek-Oberwagner, Deepak Garg, and Peter Druschel. 2016. Thoth: Comprehensive Policy Compliance in Data Retrieval Systems. In *25th USENIX Security Symposium (USENIX Security 16)*. 637–654.

[27] Dennis Fisher. 2019. Chrome and Firefox Removing EV Certificate Indicators. (August 2019). https://duo.com/decipher/chrome-and-firefox-removing-ev-certificate-indicators.

[28] French gov used fake Google certificate to read its workers' traffic 2013. French gov used fake Google certificate to read its workers' traffic. (December 2013). https://www.theregister.co.uk/2013/12/10/french_gov_dodgy_ssl_cert_reprimand/.

[29] Mark Goodwin. 2015. Revoking Intermediate Certificates: Introducing OneCRL. Mozilla Security Blog. (March 2015). `http://mzl.la/1zLFp7M`.

[30] Jacob Hoffman-Andrews. 2020. (November 2020). https://letsencrypt.org/2020/11/06/own-two-feet.html.

[31] Russ Housley, Dr. Warwick S. Ford, and Dave Solo. 1996. Internet Public Key Infrastructure. Part I: X.509 Certificate and CRL Profile. RFC 2459. (June 1996). https://datatracker.ietf.org/doc/html/draft-ietf-pkix-ipki-part1-02

[32] Ian Haken. 2017. (April 2017). https://netflixtechblog.com/bettertls-c9915cd255c0.

[33] Trevor Jim. 2000. SD3: A trust management system with certified evaluation. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 106–115.

[34] James Kasten, Eric Wustrow, and J Alex Halderman. 2013. CAge: Taming Certificate Authorities by Inferring Restricted Scopes. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*. Springer, 329–337.

[35] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. 2018. Pesos: Policy Enhanced Secure Object Store. In *Proceedings of the Thirteenth EuroSys Conference*. 1–17.

[36] Adam Langley. 2013. Enhancing digital certificate security. (2013). https://security.googleblog.com/2013/01/enhancing-digital-certificate-security.html.

[37] Adam Langley. 2013. Further improving digital certificate security. (2013). https://security.googleblog.com/2013/12/further-improving-digital-certificate.html.

[38] James Larisch, Waqar Aqeel, Michael Lum, Yaelle Goldschlag, Kasra Torshizi, Leah Kannan, Yujie Wang, Taejoong Chung, Dave Levin, Bruce M. Maggs, Alan Mislove, Bryan Parno, and Christo Wilson. 2022. Hammurabi: A Framework for Pluggable, Logic-Based X.509 Certificate Validation Policies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*.

[39] James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. CRLite: A Scalable System for Pushing all TLS Revocations to All Browsers. In *Proc. of IEEE Symposium on Security and Privacy*.

[40] Ninghui Li, Benjamin N Grosof, and Joan Feigenbaum. 2003. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)* 6, 1 (2003), 128–171.

[41] Ninghui Li and John C Mitchell. 2006. Understanding SPKI/SDSI Using First-Order Logic. *International Journal of Information Security* 5, 1 (2006), 48–64.

[42] Ninghui Li, John C Mitchell, and William H Winsborough. 2002. Design of a Role-based Trust-management Framework. In *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 114–130.

[43] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. 2015. An End-to-end Measurement Of Certificate Revocation In The Web's PKI. In *ACM Internet Measurement Conference*.

[44] Zane Ma, James Austgen, Joshua Mason, Zakir Durumeric, and Michael Bailey. 2021. Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem. In *Proceedings of the 21st ACM Internet Measurement Conference (IMC '21)*. Association for Computing Machinery, New York, NY, USA, 179–194. https://doi.org/10.1145/3487552.3487813

[45] Zane Ma, Joshua Mason, Manos Antonakakis, Zakir Durumeric, and Michael Bailey. 2021. What's in a Name? Exploring CA Certificate Control. In *30th USENIX Security Symposium (USENIX Security 21)*. 4383–4400.

[46] Joseph Menn. 2022. Web browsers drop mysterious company with ties to U.S. military contractor. (November 2022). https://www.washingtonpost.com/technology/2022/11/30/trustcor-internet-authority-mozilla/.

[47] Mozilla. [n. d.]. Revoking Trust in Two TurkTrust Certificates. ([n. d.]). https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/.

[48] Mozilla Root Store Policy [n. d.]. Mozilla Root Store Policy. ([n. d.]). https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/.

[49] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. 2005. MulVAL: A Logic-based Network Security Analyzer.. In *USENIX security symposium*, Vol. 8. Baltimore, MD, 113–128.

[50] Joel Reardon. 2022. concerns about Trustcor. (November 2022). https://groups.google.com/a/mozilla.org/g/dev-security-policy/c/oxX69KFvsm4/m/PKpJf5W6AQAJ.

[51] Quirin Scheitle, Oliver Gasser, Theodor Nolte, Johanna Amann, Lexi Brent, Georg Carle, Ralph Holz, Thomas C. Schmidt, and Matthias Wählisch. 2018. The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*. Association for Computing Machinery, 343–349. https://doi.org/10.1145/3278532.3278562

[52] Jeffrey D Ullman. 1988. Database and knowledge-base systems. (1988).

[53] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Rodrigo Rodrigues, Johannes Gehrke, and Ansley Post. 2015. Guardat: Enforcing data policies at the storage layer. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–16.

[54] Yiming Zhang, Baojun Liu, Chaoyi Lu, Zhou Li, Haixin Duan, Jiachen Li, and Zaifeng Zhang. 2021. Rusted Anchors: A National Client-Side View of Hidden Root CAs in the Web PKI Ecosystem. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1373–1387.