Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem

Zane Ma
Georgia Institute of Technology
USA
zanema@gatech.edu

James Austgen
University of Illinois at Urbana-Champaign
USA
jaa2@illinois.edu

Joshua Mason
University of Illinois at Urbana-Champaign
USA
joshm@illinois.edu

Zakir Durumeric
Stanford University
USA
zakir@cs.stanford.edu

Michael Bailey
University of Illinois at Urbana-Champaign
USA
mdbailey@illinois.edu

ABSTRACT

Secure TLS server authentication depends on reliable trust anchors. The fault intolerant design of today's system—where a single compromised trust anchor can impersonate nearly all web entities—necessitates the careful assessment of each trust anchor found in a root store. In this work, we present a first look at the root store ecosystem that underlies the accelerating deployment of TLS. Our broad collection of TLS user agents, libraries, and operating systems reveals a surprisingly condensed root store ecosystem, with nearly all user agents ultimately deriving their roots from one of three root programs: Apple, Microsoft, and NSS. This inverted pyramid structure further magnifies the importance of judicious root store management by these foundational root programs.

Our analysis of root store management presents evidence of NSS's relative operational agility, transparency, and rigorous inclusion policies. Unsurprisingly, all derivative root stores in our dataset (e.g., Linuxes, Android, NodeJS) draw their roots from NSS. Despite this solid footing, derivative root stores display lax update routines and often customize their root stores in questionable ways. By scrutinizing these practices, we highlight two fundamental obstacles to existing NSS-derived root stores: rigid on-or-off trust and multi-purpose root stores. Taken together, our study highlights the concentration of root store trust in TLS server authentication, exposes questionable root management practices, and proposes improvements for future TLS root stores.

ACM Reference Format:
Zane Ma, James Austgen, Joshua Mason, Zakir Durumeric, and Michael Bailey. 2021. Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3487552.3487813

IMC '21, November 2–4, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9129-0/21/11…$15.00
https://doi.org/10.1145/3487552.3487813

1 INTRODUCTION

TLS web server authentication functions through the web public key infrastructure (PKI). In the Web PKI, clients rely on a set of trust anchors (also called a root store) to verify web identities (e.g., DNS names and IP addresses) and their corresponding cryptographic public keys. Due to the design of the existing Web PKI, each individual trust anchor is a single point of widespread failure; a single compromised trust anchor can maliciously impersonate nearly all web identities. Thus, the security of the PKI system hinges on judicious trust: carefully deciding what roots to include and exclude from a root store. In this work, we identify the root stores utilized by popular TLS clients and how these root stores compare in their trust anchor decisions.

A wide range of applications rely on the Web PKI for TLS authentication, but it is not immediately obvious which root stores they utilize. To identify the root store providers for popular web clients, we manually investigate the top 200 most common user agent strings seen across a major CDN and definitively collect the default root stores for more than three-quarters of them. We supplement this data with additional root store providers found in other popular software, ultimately collecting the root store histories of thirteen providers. Our dataset corroborates the traditional assumption that many clients rely on the roots provided by the operating system, but we also find several libraries that provide their own root stores, such as Mozilla's NSS, NodeJS, and Java. Ordiantion analysis of these providers shows that they condense into just four independent root programs—Apple, Microsoft, Mozilla/NSS, and Java—that underlie the vast majority of TLS user agents.

Our findings demonstrate that the TLS root store ecosystem is an inverted pyramid, where hundreds of user agents rely on about a dozen root store providers that stem from three root programs (excluding Java since it is not widely used). As a result, each root program has an outsized effect on a wide range of end users. Given this reach and the fault-intolerant fragility of root stores, we diligently evaluate each root program to assess their management practices and placement of trust. We find NSS to have the best operational hygiene, followed by Apple, and then Microsoft and Java. With regards to trust decisions, we find that Apple, and Microsoft especially, place their trust in a wider range of CAs than NSS/Java. In several cases, Microsoft trusts government operated "super-CAs" that are not included by other root store programs, and

actively rejected from NSS inclusion. These comparisons suggest that Microsoft has an especially permissive root store program. Each root program serves different users and applications, so we do not expect their root stores to match or operate identically, but understanding these differences can provide insight into potential vulnerability exposure.

Possibly due to NSS's relative transparency, operational hygiene, and strictness, we find that all derivative root store providers (e.g., NodeJS, Linux distributions, Android) base their roots off of NSS. Although copying NSS simplifies root store management, this stable foundation does not necessarily imply root store security. We find that NSS derivatives do a poor job of keeping up-to-date with NSS, falling behind by almost two years in some instances. Furthermore, we observe poor NSS copying due to a fundamental design difference between NSS and derivatives: derivative root stores can only implement on-or-off root trust and cannot match NSS's capabilities for partial distrust. This results in messy trust copying, such as the re-trusting of partially distrusted Symantec roots by Debian/Ubuntu, amongst others.

Ultimately, our analysis sheds light on the modern TLS root store ecosystem, highlighting its current inverted pyramid structure, the operational practices of the foundational root programs, and existing pain points around derivative root stores. We conclude with a discussion of the future of the Web PKI, making suggestions to guide the expansion of root stores to more devices and clients as TLS deployment continues to grow.

2 BACKGROUND

The Web Public Key Infrastructure (PKI) provides a scalable solution to TLS authentication by delegating identity verification to a set of trusted certification authorities (CAs). CAs are third-party businesses, governments, and non-profit organizations that specialize in identity verification of *subscribers*, which are any entities that request a CA's services. CAs generate signed digital certificates that attest to the binding between a subscriber's identity (e.g., DNS name) and cryptographic public key. During TLS authentication, this attestation is accepted if the CA is trusted by the authenticator. In this section, we outline the role of CAs, trust anchors, and root stores, and refer the reader to [87] for a broader overview of TLS.

All user agents that utilize the Web PKI rely on one or more trusted "root" CAs that comprise its trust anchor store, or root store. Common expectations for CA behavior, such as secure identity verification procedures and strict operational security practices (e.g., hardware-secured cryptographic keys), are codified by the CA/Browser Forum in its Baseline Requirements (BRs). The BRs standardize CA requirements, but stop short of prescribing how root programs should decide which CAs to trust. This role is filled by the custom CA inclusion and removal policies of each root store operator. For instance, Mozilla runs a relatively rigorous policy that requires roots to publicly disclose unconstrained intermediates in its Common CA Database (CCADB) [99], while other root stores do not. CAs demonstrate their policy compliance through Certificate Policies (CP) and Certification Practice Statements (CPS), and many root stores require regular CA audits to confirm policy adherence.

A CA's digital identity is a public-key and name mapping, and root stores typically represent these identities as X.509 digital certificates. In addition to storing identity, these certificates also contain constraints on the functionality of a trust anchor. For example, all X.509 certificates contain a validity period that limit the duration of a root CA's utility. Additionally, the Key Usage (KU) and Extended Key Usage (EKU) X.509 extensions specify the permitted roles (e.g., certificate signature, key agreement) and trust purposes (e.g., TLS server / client authentication) of the certificate. As suggested by the presence of EKU, the Web PKI is used for more than just TLS server authentication. CAs that are used for TLS server authentication often issue certificates for other common purposes, namely email signatures (i.e., S/MIME) and code signing. The BRs only apply to publicly-trusted TLS server certificates, and the identity verification requirements and procedures for other trust purposes differ. A root CA that is trusted for one form of identity verification is not necessarily qualified or trusted for other forms. For this paper, we focus on TLS server authentication certificates that are primarily used for HTTPS but can also be used for secure email transport (STARTTLS).

CAs dictate the constraints found within a certificate, but end entities such as browsers can also specify their own restrictions. For instance, Mozilla's Network Security Services (NSS) and Microsoft root stores contain additional trust constraints, external to their trusted root certificates. The BRs only permit CA-specified EKU trust purposes for intermediate CA certificates, not roots, so root programs without their own external trust policies have poor visibility into what their roots are trusted for.

3 ROOT STORE PROVIDERS

The first challenge with understanding the landscape of HTTPS trust anchor stores is to determine who provides them. Operating systems provide a system-wide store, but HTTPS libraries and HTTPS clients can supplant these system roots by shipping their own root store. In order to determine the providers of root stores used by TLS clients, we took a two pronged approach: looking at popular user agents in the wild and investigating well-known operating systems, TLS libraries, and TLS clients.

As a starting point, we collected the top 200 User Agent HTTP headers (with bots removed) seen during a 10 minute sample of traffic from a major CDN on April 7, 2021 around 18:00 UTC. We were unable to acquire multiple/longer samples, and cannot test the representativeness of our time period. However, we note that a prior study [94] found that "broadly speaking, this list [of prevalent UAs] is stable over time." Table 1 groups User Agents (i.e., client software) with operating system, and we use this data as a guide for collecting root stores in common use on the web. In total, we collected the root stores for 77% of the top 200 UAs. The 23% of user agents for which we have unknown or no coverage consist of ChromeOS, less common browsers (e.g., Yandex, Samsung Internet), custom applications making API calls, or clients that cannot be identified by their User Agent header. We exclude Yandex and Chrome on ChromeOS because the source history is not publicly available.

We supplement the popular user agent dataset by expanding our data collection process (Appendix A) to consider a total of nine popular mobile and desktop operating systems, which all

OS/User Agent	# versions	Included?
Android		
Chrome Mobile	48	yes
Samsung Internet	2	no
Android	3	no
Firefox Mobile	1	yes
Chrome Mobile WebView	1	no
Chrome	1	yes
Windows		
Chrome	23	yes
Firefox	7	yes
Electron	6	yes
Opera	4	yes
Edge	4	yes
Yandex Browser	3	no
IE	3	yes
iOS		
Mobile Safari	18	yes
WKWebView	4	yes
Chrome Mobile iOS	2	yes
Google	2	no
Mac OS X		
Safari	15	yes
Chrome	14	yes
Firefox	2	yes
Apple Mail	1	no
Electron	1	yes
ChromeOS		
Chrome	8	no
Linux		
Chrome	2	no
Safari	1	no
Firefox	1	yes
Samsung Internet	1	no
Unknown		
okhttp	3	no
Unknown	2	no
CryptoAPI	1	no
API Clients	*16*	*no*
Total included	**154 (77.0%)**	

Table 1: Major CDN Top 200 User Agents—We collect root store history for at least 77% of popular clients.

provide a root store for applications running on each platform. We also examined nineteen widely used TLS libraries and found that only three (NSS, NodeJS, Java Secure Socket Extensions) provide a root store that application developers may use. The remaining TLS libraries either default to the base platform/OS root store or are configurable at build time. Finally, we looked into twelve common HTTPS clients and web browsers and find that only 360Browser, Firefox, and Chrome do not rely on system root stores and ship their own. Table 2 summarizes our final root store dataset, and we further describe each root store provider below.

NSS/Firefox/Mozilla Mozilla's Network Security Services (NSS) is a set of libraries that provide secure client-server communications. Mozilla develops and uses NSS solely for its Firefox web browser, Thunderbird email client, and "other Mozilla-related software products" [57]. Since 2000, NSS has maintained a trust anchor store through its `certdata.txt` file. This file follows the PKCS#11 format and contains a list of two types of elements: certificates and trust objects. Certificates contain raw certificate data and some extracted fields. Trust objects contain 1) *trust anchor identifiers* (i.e., issuer name, serial number, and SHA1/MD5 hashes of a certificate) as well as 2) *trust details*, which include the trust purpose (i.e., server authentication HTTPS, email protection S/MIME, code signing) and level (i.e., trusted, needs verification, or distrusted). This trust context is solely determined by NSS maintainers through independent due diligence, such as audit review and discussion with the PKI community.

Unfortunately, not all of Mozilla's trust anchor policies are fully encapsulated within `certdata.txt`, due to their complexity and technical considerations [33]. For instance, special constraints for the Turkish government CA are implemented in C / C++ code. Also the distrust of Symantec was partially implemented in code to whitelist subordinate CAs of Symantec that were independently operated. Mozilla also manages EV trust outside of `certdata.txt` [33]. As discussed in Section 3.1, we do not account for these nuanced modifications when performing broadly-scoped analysis, but we do consider them when examining specific roots.

NSS is the most well maintained trust anchor store for HTTPS (and TLS server auth) by several measures. First, NSS has the most transparent root inclusion / removal process. NSS abides by the Mozilla Root Store policy, which requires maintainers to work with the community [50, 51] to solicit feedback on all root CA inclusion/removal proposals and on policy improvements for the Mozilla Root Store. NSS also monitors CA issues through its public bug tracking system and presents evidence for actions taken against problematic CAs [36, 39, 40]. Second, NSS holds CAs to a relatively high standard through its strict root store policy. CA issues frequently first surface through NSS bug reports. As a result, NSS is also the most responsive root store, often mitigating CA incidents ahead of other root stores. For instance, Mozilla lead the community discussion of DarkMatter's trustworthiness as a CA [108], and uncovered WoSign's surreptitious ownership of StartCom [100]. We refer to NSS/Mozilla interchangeably in this manuscript.

Microsoft Microsoft updates the root certificates for its Windows operating systems via Automatic Root Updates (partially supported as early as Windows XP SP2 [76]) through which Microsoft ships `authrootstl.cab`. This file decompresses to `authroot.stl` and contains a list of trust anchors and their Microsoft-specific OIDs, which specify restrictions on each trust anchor. These restrictions define the purposes for which a certificate is trusted or distrusted, as well as other more nuanced trust, as discussed later in Section 5.3. Full certificates are not included in `authroot.stl`, but they can be downloaded from Microsoft by SHA1 hash through a separate URL. This study uses an open-source archive of `authroot.stl` and associated certificates [107].

Apple Since at least 2005 [75], Apple has managed its own root certificate program to support products such as "Safari, Mail.app, and iChat." More recently, this root store has supported both macOS and iOS product lines. Apple stores trust anchors in keychain files that can contain a wide range of credentials (e.g., passwords, private

Root store	From	To	# SS	# Uniq	Data source	Details
Alpine	2019-03	2021-04	42	7	docker [46]	`/etc/ssl/cert.pem` or `/etc/ssl/ca-certificates.crt`
AmazonLinux	2016-10	2021-03	43	15	docker [47]	`ca-trust/extracted/pem/tls-ca-bundle.pem` aggregate file of root certs
Android	2016-08	2020-12	14	7	source code [26]	List of root certificate files.
Apple	2002-08	2021-02	109	43	source code [69]	Both macOS and iOS. `certificates/roots` directory of files
Debian	2005-05	2021-01	39	29	source code [45]	`/etc/ssl/certs` and `/usr/share/ca-certificates`, directory of cert files
Java	2018-03	2021-02	7	7	source code [62]	`make/data/cacerts` JKS file that has migrated over time
Microsoft	2006-12	2021-03	86	70	update file [107]	`authroot.stl` updates roots, trust purpose, addl. constraints
NodeJS	2015-01	2021-04	16	11	source code [59]	`src/node_root_certs.h` list of certificates
NSS	2000-10	2021-05	225	63	source code [58]	`certdata.txt` stores roots, trust purpose, additional constraints
Ubuntu	2003-10	2021-01	38	29	source code [73]	`/etc/ssl/certs` and `/usr/share/ca-certificates`, directory of cert files

Table 2: Dataset—Root store history of 619 total snapshots (SS) for ten root providers: seven OS, three library.

keys, etc.) in addition to root certificates. While recent versions of the keychain format are capable of specifying specific key usages (`kSecTrustSettingsKeyUsage`), specific usage restrictions are not provided by default. We collect roots from Apple's open source repository [69].

Linux distributions Most Linux distributions derive their trust anchor stores from NSS. However, rather than use NSS's custom `certdata.txt` file, they express their trust through a list of X.509 certificates stored in a menagerie of directories. This format for trust anchor stores omits the trust purposes that are specified by NSS/Microsoft. To account for this discrepancy, recent versions of AmazonLinux, Fedora, and others provide additional purpose-specific root stores[1] that distinguish between TLS server authentication, S/MIME email signing, and code signing use cases. For this study, we only consider TLS server authentication certificates when the distinction is available. Although they rely on NSS, Linux trust anchor stores are updated manually and may make custom modifications to `certdata.txt`. To account for this possibility, we either run the build process to extract accurate root store information, or we collect data from a pre-built, officially distributed Docker image.

Android Android maintains its own trust anchor store [26]. It consists of three root directories: general purpose, Google Services, and Wi-Fi Alliance (WFA). We collect the general purpose roots only. Although the Android root store repository has been active since 2008, Android version tags have only been applied since 2015, so we only have definitive snapshots after that date.

Chrome Chrome installations traditionally inherited the operating system trust store (except on ChromeOS or for EV) with its own specialized control. For example, to protect its users against the distrusted CA Symantec, Chrome implemented bespoke CA distrust policies across all platforms besides iOS[2]. In late 2020, Google announced their transition to its own TLS root store [41] to provide a consistent experience for all of its users. However, as of May 2021, the transition is still in-progress, and we exclude it from this study.

Java Oracle, the developer of Java, operates a root program [52] to provide Java developers with a default set of root CAs for TLS server authentication, email signing, and code signing. We measure these trusted CAs through OpenJDK's source repository. These CA certificates are typically stored in a Java-specific JKS file, which we parse using Java's `keytool` utility. Java's default root store does not include additional trust contexts or restrictions.

NodeJS NodeJS provides a compile flag to trust system root stores, but by default, it ships a file that contains trusted root certificates.

Opera Opera maintained its own root store until 2013 [66], when it switched to adopting NSS's root store and Chrome's EV store. Opera does not provide its software open source, so we do not include it in our dataset. Opera migrated to Chromium in 2013 and now utilizes system root stores.

Electron Electron is an application development framework used by applications such as Slack, Discord, Skype, etc. that combines NodeJS and Chromium to allow developers to build applications using only web technologies: JavaScript, HTML, and CSS. Depending on the networking library used [79], Electron can rely on either Node's root store, or the system root store, which Chromium defaults to.

3.1 Data Collection & Limitations

The root store providers described above manage and publicly release[3] their trust anchors in different formats. We parse these formats and consolidate them into a single database. For each root store provider, we store *snapshots*, which represent a root store at a single point in time. Each snapshot is a collection of *trust entries* that include a certificate along with any additional trust/distrust constraints (e.g., as provided by NSS and Microsoft). This study only examines root stores, and does not evaluate certificate chains or intermediate certificates, which are complicated by cross-signing, CRL/OCSP certificate revocation, and other client-specific methods such as Mozilla's OneCRL and Chrome's CRLSets.

Our dataset and methodology have a few limitations. First, the dates for each root store snapshot do not always reflect the earliest release date of each root store; instead, they should be viewed as approximations. We take a best-effort approach to collecting the root stores for a wide range of OSes and TLS software, and as as result, our data collection represents different stages of root store deployment. For some root store providers, we can collect

[1] Whether applications utilize these purpose-based roots is beyond the scope of this study.
[2] Apple prohibits custom root policies on iOS.
[3] Notable exceptions with no reliable root history include ChromeOS and Fedora/-CentOS. The latter releases Docker images, but with inconsistent timestamping and versioning.

the source code repositories, which provide release tags that are a proxy for release dates. For others, we can only collect pre-built Docker images or root store update files, which may not correspond perfectly with actual release dates. In order to compare the root store dates derived from different means, we treat snapshot dates as a rough approximation, and only make coarse-grained comparisons between them, on the order of months or years.

Second, the data we collect represents default values and may not reflect customized root store deployments. While no prior studies have comprehensively measured root store deployments in the wild, some have suggested that root stores may be altered by cellular carriers [109] and locally installed AV / monitoring software [90]. We recognize that our dataset represents only the default root stores provided by popular OSes and TLS software. However, we have not discovered any reports of manual trust anchor removal from default root stores[4]. This fact, coupled with the fault-intolerant nature of the TLS PKI means that the results from our study are likely a lower bound on real-world deployments.

Third, certificate chain validation is complex, and understanding root stores is only part of the overall process that involves chain-building and bespoke trust restrictions embedded in code. A root certificate's inclusion in a trust anchor store does not guarantee that it is trusted. From our experience looking through TLS library code, additional modifications to root store trust typically address exceptional cases, rather than commonplace scenarios. When we discuss specific root store inclusions/removals in Sections 5 and 6, we make a best-effort attempt to account for any trust logic external to the root store itself. For example, Apple utilizes a custom revocation mechanism that downloads over-the-air updates from `valid.apple.com`, and we note when this extends to questionable roots.

4 ROOT STORE FAMILIES

Although OSes and some libraries/clients ship their own root stores, they are not necessarily independent. Properly managing a root store takes significant, sustained effort, and not all TLS software developers have the capacity to manage a root store. Instead, some root store providers derive their roots from other sources, making identical copies or bespoke modifications. For instance, many Linux distributions rely on NSS as the foundation of their root stores. Unfortunately, not all root stores are open source and can be traced directly to an independent source (e.g., NSS) through documentation. Some of our data sources are pre-built software (e.g., docker images of AmazonLinux / Alpine), which lack transparent root store provenance. To develop a general mechanism for determining the interrelatedness/lineage of root stores, we take inspiration from community ecology. We perform ordination analysis to visualize the relationship of communities (collection of trust anchors in a root store) across different sites (OS/library/client).

We collect root store histories and perform multidimensional scaling (MDS) to cluster root store providers based on the pairwise Jaccard distance between each root store community over time. MDS is a dimensionality reduction technique where the lower dimensional representation preserves intra-object distances as well as possible. We use the stress majorization variant of metric MDS as implemented by Python's `sklearn` library [104].

From Figure 1, we can see four clusters emerge from the root store providers in our dataset. These correspond (from left to right) with Microsoft, NSS/Linux/NodeJS, Apple, and Java. Each cluster reflects a *family* of root providers that rely on a single independent root program. The clusters are disjoint and do not overlap (excluding three Apple and one Java outlier described below), which indicates that as each root store family has evolved, they have not converged or diverged with other root programs. Only the NSS cluster contains derivative root stores (Android, Linux distributions, and NodeJS) that copy the NSS root store. However, we also observe derivative snapshots that do not completely overlap with NSS snapshots. This suggests not all NSS derivatives make perfect copies; some make custom modifications, which we explore further in Section 6.

We observed four outliers in our cluster analysis, all located between the NSS and Microsoft clusters. These outliers all occur when substantial changes occur before and/or after a given snapshot. For example, one Java outlier from August 2018 occurs due to the removal of 9 roots (3 of which were unique to Java), and the addition of 21 roots, which is a total of 30 (37.5%) changed certificates between relatively small Java snapshots. The subsequent Java version removed 6 roots and added 2 new roots. Similar incidents occur for Apple's October 2011 snapshot (10 changed roots), February 2014 snapshot (67 changed roots), and September 2018 snapshot (19 changed roots). Due to the lack of root program transparency, we can only examine what roots changed and try to infer why these large changes occurred. For example, in the most prominent outlier (Apple February 2014) we see 9 root removals, and the addition of a large number of diverse CAs after nearly one-and-a-half years of Apple root store stagnation. This outlier is either a lapse in Apple's open-source repository data, or an intentional delayed update in one large batch.

The outliers are also exaggerated by their visual location in the MDS plot. Unfortunately, the high dimensionality (243 dimensions) of pairwise snapshot distances—the reason we use MDS—impedes us from definitively explaining the location of the outliers between Microsoft and NSS in two dimensional space. Instead, we provide an intuitive rationale: these large deviation outliers are sufficiently distanced from other Java/Apple snapshots that the large number of Microsoft/NSS-like snapshots causes MDS to prioritize the preservation of these high-volume long range interactions over the preservation of fewer, but closer interactions with Apple snapshots. Put another way, placing the Apple outliers closer to Apple (i.e., between NSS and Apple) would misrepresent the distance between the outliers and all NSS-like snapshots and all Microsoft snapshots, which account for 88% of all snapshots.

To understand the utilization of root stores in the wild, we traced the top 200 user agents to their root store family and found that NSS (34%), Apple (23%), and Windows (20%) together account for a majority of the user agents. Java is not linked to any of the top user agents. As shown in Figure 2, the root store ecosystem is an inverted pyramid, with a diversity of user agents relying on a centralized foundation of three root programs. This structure inflates the importance of CA trust decisions made by each foundational root program, which we examine below.

[4]Chrome/Firefox apply their own restrictions on top of root stores, but do not modify them.

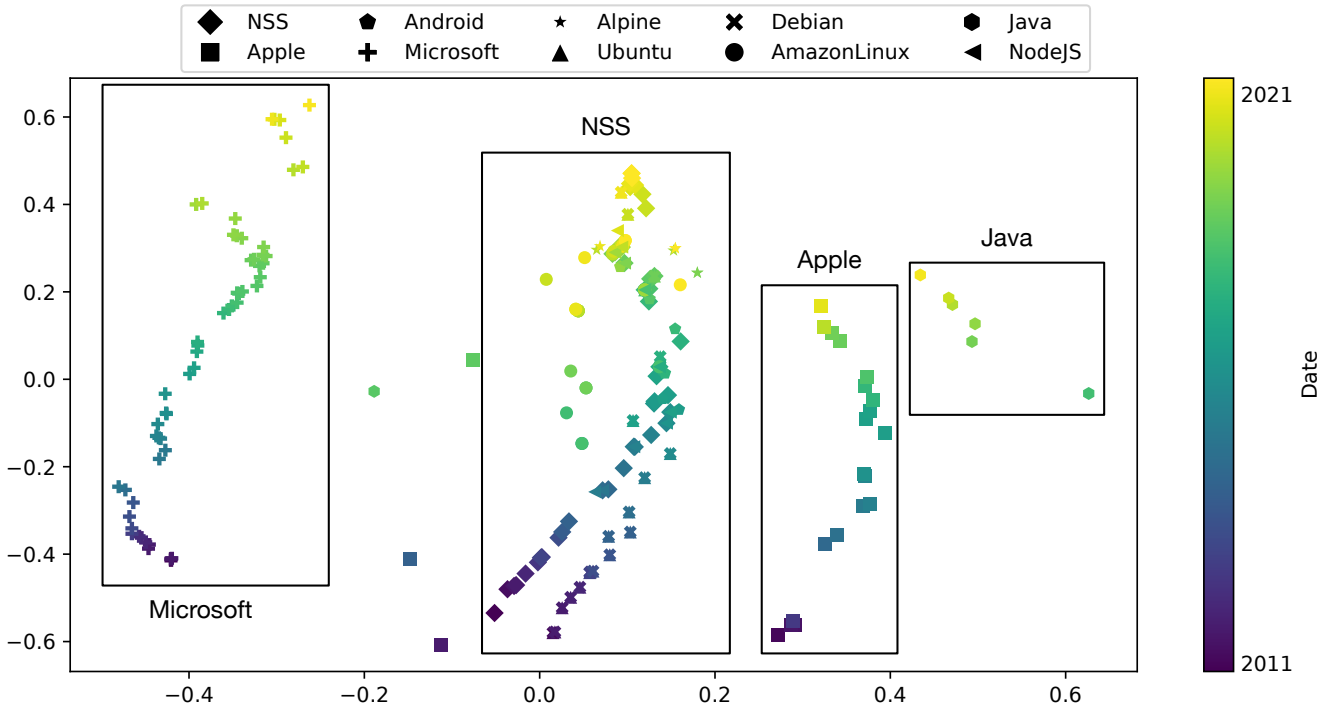

Figure 1: Root Store Similarity—Performing MDS on the Jaccard distance between root store providers from 2011–2021 illustrates four distinct clusters of roots. From left to right: Microsoft, NSS-like, Apple, Java.

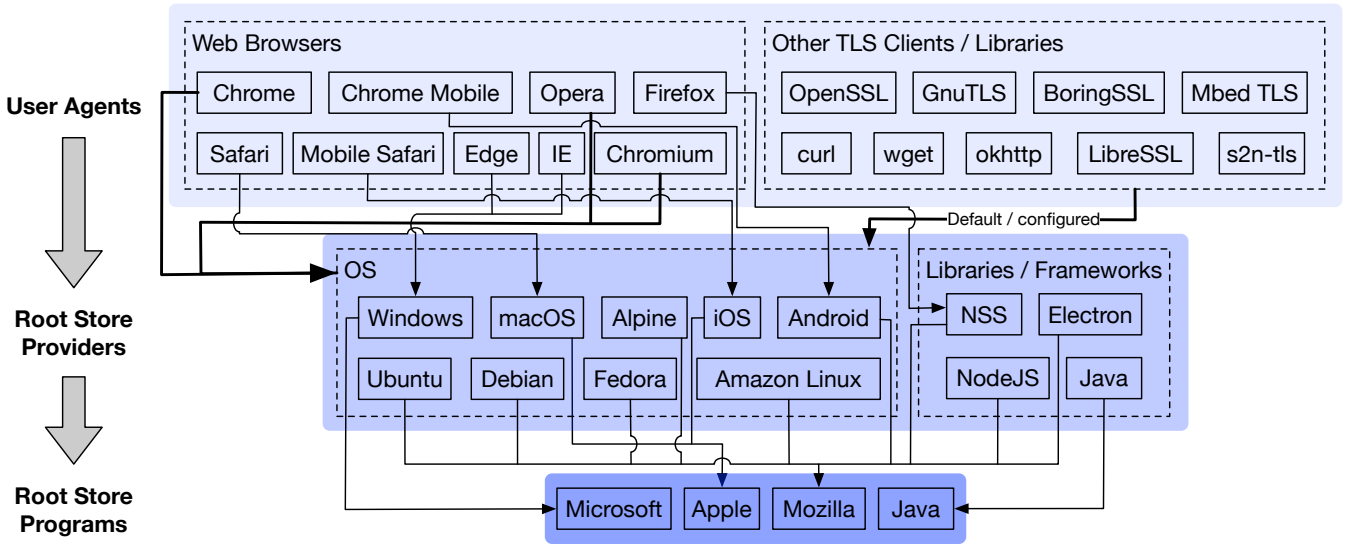

Figure 2: Root Store Ecosystem—The TLS root store ecosystem is an inverted pyramid, with a majority of clients trusting one of four root families.

5 COMPARING ROOT STORES

In this section, we evaluate the four independent root store programs used for TLS server authentication: Apple, Java, Microsoft, and NSS. We compare their security-relevant hygiene, response to major CA distrust events, and investigate differences in CA trust. In doing so, we aim to better understand the operational behavior of each root store and gain insight into their observed differences.

Root store	Avg. Size	Avg. Expired	MD5	1024-bit RSA
Apple	152.9	2.9	2016-09	2015-09
Java	89.4	1.3	2019-02	2021-02
Microsoft	246.6	9.9	2018-03	2017-09
NSS	121.8	1.2	2016-02	2015-10

Table 3: Root store hygiene—The average number of total and expired roots in each root store snapshot and removal dates for trusted MD5/1024-bit RSA certificates.

5.1 Root store management

As a proxy for responsible root store management, we examine three metrics (Table 3): removal of roots with MD5-based signatures, removal of roots with 1024-bit RSA keys, and removal of expired root certificates. Apple and NSS were the most proactive in purging 1024-bit RSA and MD5 certificates, in 2015 and 2016, while Microsoft took 2 additional years, and Java even longer. On the other hand, NSS and Java have fewer expired roots present in each root store update than Apple and especially Microsoft, which averages nearly 10 expired roots. Microsoft does manage a larger root store, but this is not proportional to the increased expirations. Our results suggest that NSS exhibits the best root store hygiene, followed by Apple, and then Java/Microsoft.

5.2 Exclusive Differences

To better quantify the differences between each root store family, we characterize the root CAs that are unique to each. Appendix B displays the unique, most recently trusted roots for each root store that have never been trusted for TLS server authentication by any of the other independent root programs. For each root, we look for an NSS inclusion request as an additional data source about a given root and its reason for requested inclusion. We also identify the CA operator for each root by examining CCADB and the certificate itself. The unique roots for each store are described below.

NSS The only NSS root not trusted by other root programs is a newly included Microsec root that uses elliptic curve cryptography (ECC). This exclusive root does not indicate NSS-only trust in Microsec; rather, the new root accompanies an existing Microsec root that is already trusted by NSS, Apple, and Microsoft.

Java Java operates a relatively small root store that includes substantially fewer roots than the other three root programs. No Java-exclusive root trust is observed.

Apple The thirteen Apple-exclusive roots can be categorized into three broad categories. First, six roots are trusted by Microsoft or NSS, but only for email. Apple has the technical mechanisms to restrict the trust purposes for each root, but it does not appear to provide default policies that specify which roots should be used for which purposes. Second, five roots are controlled by Apple's CA and utilized primarily for Apple specific services, such as FairPlay and Developer ID code signing. This is an expected divergence in trust from other root programs since they do not participate in proprietary Apple software and protocols. Finally, we discover two roots that are actively distrusted by either Microsoft or NSS. Apple's trust in the Certipost root is likely benign, since the CA requested revocation in NSS "solely because they no longer issue SSL/TLS server certificates." Apple's inclusion of a Government of Venezuela root is more questionable. This root was rejected from NSS due to the CA's position as a super-CA [71] that acts as a trust-bridge to large numbers of independent subordinate CAs, which each have the capability to issue trusted certificates for any TLS server identity[5]. One such subordinate CA, PROCERT, gained entry into NSS in 2010, but was subsequently removed after repeated transgressions [38]. This issuer was also trusted by Microsoft, but only for email, until it was blacklisted in 2020. Apple's custom revocation system has blocked this root, but its inclusion in the set of shipped trust anchors presents an opportunity to clean up untrusted roots.

Microsoft Microsoft contains 30 exclusive root certificates. Eleven of these roots attempted and failed the NSS inclusion process, either due to NSS rejection (7 roots) or CA abandonment (4 roots) after critical review. Three of these roots belong to national governments (Brazil, Korea, Tunisia), and two out of three were rejected from NSS due to secret or insufficiently disclosed subCAs. Worryingly, Microsoft also trusts a unique root belonging to AC Camerfirma, which was removed from NSS in May 2021 due to a long-running list of misissuances. Microsoft's inclusion of these roots indicates a lower standard for trust in root CAs. The remaining 19 Microsoft-exclusive roots reflect a more innocuous collection of CAs with ongoing NSS inclusion evaluation (6 roots), recently accepted in NSS (3 roots), trusted by Apple/NSS through cross-signing (2 roots), minimal Certificate Transparency presence (6 roots), a WiFi Alliance root for automatic WiFi roaming, and a Government of Finland root. For more details, see Appendix B.

Takeaways Root programs serve different applications and users, and thus, their root CA trust decisions will likely differ. By examining the exclusive differences between root programs, we highlight some of their major policy distinctions. Microsoft appears to have a stronger tolerance for national government super-CAs, while Apple utilizes its root program to manage roots for TLS server authentication, email signing, and its own proprietary services. Future work can determine the security consequences of these uniquely trusted roots and evaluate the overall performance (scale and security, see Section 7) of each root program.

5.3 Trusting NSS removals

Another measure of a root store's responsible management is its agility and responsiveness to root CA incidents. Since NSS provides the only transparent mechanism for CA issue tracking, we catalog all NSS removals after 2010 , track the Bugzilla bug report, and group the issue into one of three severities: low, medium, high. Low severity issues consist of routine removal of expired roots or removal of roots at the request of the CA, typically due to cessation of operation. Medium severity removals are driven by Mozilla due to non-urgent security concerns. High severity indicates a Mozilla-prompted removal due to urgent security concerns. These high and medium severity removals are shown in Appendix C. Although

[5]Super-CAs are not prohibited, but NSS requires sub-CAs to be audited and accounted for essentially as a stand-alone CA.

Root store	# Certs	Trusted until	Lag (days)
DigiNotar [101]		**2011-10-06**	
Microsoft	1	2011-08-30	-37
Apple	1	2011-10-12	6
Debian/Ubuntu	1	2011-10-22	16
CNNIC [78]		**2017-07-27**	
Apple	2	2015-06-30	-758
Android	1	2017-12-05	131
Debian/Ubuntu	2	2018-04-09	256
NodeJS	2	2018-04-24	271
AmazonLinux	2	2019-02-18	571
Microsoft	2	2020-02-26	944
StartCom [113]		**2017-11-14**	
Debian/Ubuntu	3	2017-07-17	-120
Microsoft	2	2017-09-22	-53
Android	3	2017-12-05	21
NodeJS	3	2018-04-24	161
AmazonLinux	3	2019-02-18	461
Apple	3	1 root still trusted	1,175+
WoSign [113]		**2017-11-14**	
Debian/Ubuntu	4	2017-07-17	-120
Microsoft	4	2017-09-22	-53
Android	4	2017-12-05	21
NodeJS	4	2018-04-24	161
AmazonLinux	4	2019-02-18	461
PSPProcert [38]		**2017-11-14**	
Debian/Ubuntu	1	2018-04-09	146
NodeJS	1	2018-04-24	161
AmazonLinux	1	2019-02-18	461
Certinomis [37]		**2019-07-05**	
NodeJS	1	2019-10-22	109
Alpine	1	2020-03-23	262
Debian/Ubuntu	1	2020-06-01	332
Android	1	2020-09-07	430
AmazonLinux	1	2021-03-26	630
Apple	1	2021-01-01*	577
Microsoft	1	Still trusted	607+

**Revoked via valid.apple.com at unknown date.*

Table 4: High severity removals—Comparison of root store responses to high severity NSS removals.

Mozilla provides a Removed CA Report [64], this data misses 92 removals (mostly due to expiration or CA removal request) found in our manual analysis. It also includes two incomplete "removals", where a CA is distrusted for email/code signing but remains trusted for TLS, that our dataset does not capture. We have notified Mozilla of this discrepancy.

Table 4 shows the responsiveness of different root stores to high-severity removals. We do not include medium and low severity removals, since root store inclusion / removal decisions are highly contextual to different root stores, and we do not expect all root stores to respond to lower severity removals. We do not discuss all trust actions (e.g., revocations, cross-signing, etc.) for each incident, only examining the relevant root store details, and provide references to more detailed descriptions.

Diginotar In 2011, attackers gained access to DigiNotar private keys and forged trusted certificates for high-profile websites [82], leading to the most serious PKI security exposure of the last decade. Microsoft, Apple, and Mozilla swiftly removed DigiNotar's root certificate from their root stores. The observed differences in removal time reflect the nature of our data sources: we detect immediate Microsoft updates, but only detect removal from Apple/NSS in the next published root store snapshot. In reality, Mozilla pushed an update on August 29, 2011 [102], and Apple followed suit on September 9 [77]. Although the up-to-the-hour response delay in such incidents is important, our dataset lacks the resolution (see Section 3.1) to provide more fine-grained analysis.

CNNIC In 2015, Google discovered a Mideast Communication Systems (MCS) intermediate issued by China Internet Network Information Center (CNNIC) issuing forged TLS certificates. Beyond the questionable ethics of the incident, the intermediate certificate's private keys were installed on a firewall device, exposing it to potential compromise. In response to this situation, Chrome, Mozilla, and Microsoft immediately revoked the MCS intermediate certificate, and Mozilla implemented partial distrust of CNNIC roots in code, only trusting certificates issued before April 1, 2015 [42]. Apple took an alternate approach—they removed the CNNIC root in 2015, significantly before other root programs, but whitelisted 1,429 leaf certificates [1]. This accounts for Apple's preemptive removal of CNNIC roots. Microsoft took the most permissive approach and continued to trust CNNIC roots until 2020.

StartCom / WoSign In 2016, Mozilla discovered that the CA WoSign was secretly backdating SSL certificates to circumvent Mozilla's deadline for halting SHA1 certificate issuance [113]. Further, Mozilla discovered that WoSign had stealthily acquired another CA StartCom and found evidence that StartCom was utilizing WoSign's CA infrastructure. In response, Mozilla (and Chrome) implemented code changes to partially distrust WoSign/StartCom certificates in late 2016, eventually removing the seven roots in 2017. Microsoft only began to partially distrust StartCom / WoSign nearly a year later. Apple never included WoSign roots directly, but revoked their trusted cross-signed intermediates. Surprisingly, Apple still trusts one of the three StartCom roots[6], despite knowledge of WoSign ownership and evidence of shared issuance.

Procert Procert was never included in Apple, Microsoft, or Java root stores, and not subject to removal.

Certinomis Amongst other transgressions, Certinomis cross-signed a StartCom root *after* StartCom had been distrusted, effectively creating a new valid trust path for StartCom. Furthermore, Certinomis delayed disclosure of these cross-signs by 111 days. Apple has revoked this root but has not removed it, while Microsoft continues to trust this root certificate.

[6]Two StartCom roots were revoked via `valid.apple.com`, but not removed.

Takeaways Although root programs make independent root trust decisions based on their own contexts, we expect them to remove trust in high severity root CA removals. We observe a range of different response times, response mechanisms (e.g., remove root and whitelist leaves, revoke root, partially distrust root, etc.), and even a lack of response for some root programs. The heterogeneity of responses suggests room for improvements such as the convergence of trust mechanisms or more prompt removal procedures.

6 NSS DERIVATIVES

In NSS documentation, Mozilla states: "Mozilla does not promise to take into account the needs of other users of its root store when making such [CA] decisions...Therefore, anyone considering bundling Mozilla's root store with other software needs to...maintain security for their users by carefully observing Mozilla's actions and taking appropriate steps of their own" [57]. Despite the prevalence of NSS-derivative root stores, Mozilla's root store is explicitly not intended to be a one-size-fits-all solution for TLS authentication trust. In this section, we seek to understand the root management practices of NSS derivatives and how they influence overall security. We first examine the frequency and delay of updates to understand the risks that different NSS derivatives expose their users to. We then look at the fidelity with which root stores copy NSS, including the degree to which trust purpose restrictions are applied. We ultimately detail the custom modifications that individual root store providers make to best serve their users.

6.1 Update Dynamics

To understand the update dynamics of NSS derivatives, we first need to link specific derivative root store snapshots to the NSS version that they copy. Because NSS derivatives don't always make exact copies of the NSS root stores, we cannot look for exact root store matches. Instead, we use Jaccard set distance and find the closest NSS version match for each derivative root store snapshot. Figure 3 depicts the evolution of NSS and its derivatives over time, only including *substantial versions* that introduce changes to TLS trusted roots. Because software development often runs in parallel (e.g., maintenance support for v1, and new development for v2), we only consider *mainline* versions of each derivative that reflect the highest version at a given point in time.

To quantify derivative staleness, we integrate the area between NSS and each NSS derivative root store. This yields a "substantial version-days" measure where versions are not sequential. We then normalize these version-days over time to determine the average substantial version staleness for each root store. The data suggests that Alpine Linux, which has the shortest and most recent data collection range, adheres closest to NSS updates. On the other hand, Amazon Linux exhibits an average staleness of more than four substantial versions. Furthermore, Amazon Linux and Android are always stale—even when they update, the updated root store is already several months behind. This hints at prolonged deployment cycles that exceed NSS's relatively frequent updates. Each tick in Figure 3 represents a mainline version update for each derivative, which suggests that some derivative version updates ignore potential NSS updates, especially for Amazon Linux and Alpine.

6.2 Derivative Differences

Figure 4 depicts the root store differences between NSS and NSS derivatives (mainline versions) over time. We find that all derivatives in our dataset make bespoke modifications to the NSS root store, and we categorize and describe the reasons for these changes below.

Symantec distrust In addition to poor update adherence to NSS's root store, one shortcoming of NSS derivatives is their trust store design, which lacks a mechanism for external restrictions on root certificates. Such a mechanism could provide the ability to trust roots for specific purposes (e.g., TLS server auth, email signing, code signing) or provide gradual distrust, rather than a single on-or-off toggle. To present the practical implications of these issues, we look at the distrust of Symantec, which required nuanced trust mechanisms to handle correctly. Symantec's distrust also amplified existing pain points due to its scope—Symantec was the largest CA by issuance volume at the time of its distrust.

Beginning in late 2018, Firefox, independent of NSS, implemented a gradual distrust of Symantec by adding custom validation code [80] to distrust subscriber certificates issued after a certain date. In 2020 [72], NSS version 53 implemented partial distrust of twelve Symantec (now owned by DigiCert) roots through the new restriction *server-distrust-after* in `certdata.txt`. This had the effect of partitioning Symantec subscriber certificates into two parts: certificates that were still trusted until expiration, and certificates that were no longer trusted. However, none of the NSS derivatives had such a mechanism and were forced to choose between prematurely removing all trust in Symantec roots, or retaining full trust in Symantec roots. From our dataset, Alpine and Android have not yet upgraded beyond NSS version 48 and have postponed Symantec distrust. NodeJS skipped the Symantec distrust update in NSS version 53 and has continued to apply subsequent NSS updates. Unfortunately, version 53 also included the immediate removal of two other roots: TWCA due to Mozilla policy violations and SK ID Solutions due to CA request. These roots are preserved in NodeJS.

Ubuntu/Debian took the alternate approach and—just a few days after NSS implemented partial distrust in version 53—removed the Symantec roots. Surprisingly, they did not apply all changes introduced in version 53, and only removed eleven out of twelve Symantec roots, curiously retaining GeoTrust Universal CA 2 [32]. Unfortunately, this premature full distrust led to so many user complaints [31] that Debian re-added Symantec distrusted roots. This incident not only broke applications using the system root store for TLS server authentication, it also broke (and provided anecdotal evidence of) applications that relied on these roots for code-signing and timestamping purposes, such as Microsoft's .NET package manager NuGet [88]. This is a clear example of root store misuse, since the source of Debian/Ubuntu's root store is NSS, which only trusts CAs for TLS auth and email signing purposes. Further, Ubuntu/Debian now only include NSS roots that are trusted for TLS authentication (see Email signing below).

Non-NSS roots Ubuntu, Debian, and Amazon Linux include roots that have never been in NSS. Amazon Linux includes a single non-NSS root (3f9f27d: Thawte Premium Server CA) that it trusts from October 2016 until December 2020, just before its expiry. This root is part of the Thawte CA (acquired by Symantec, then DigiCert),

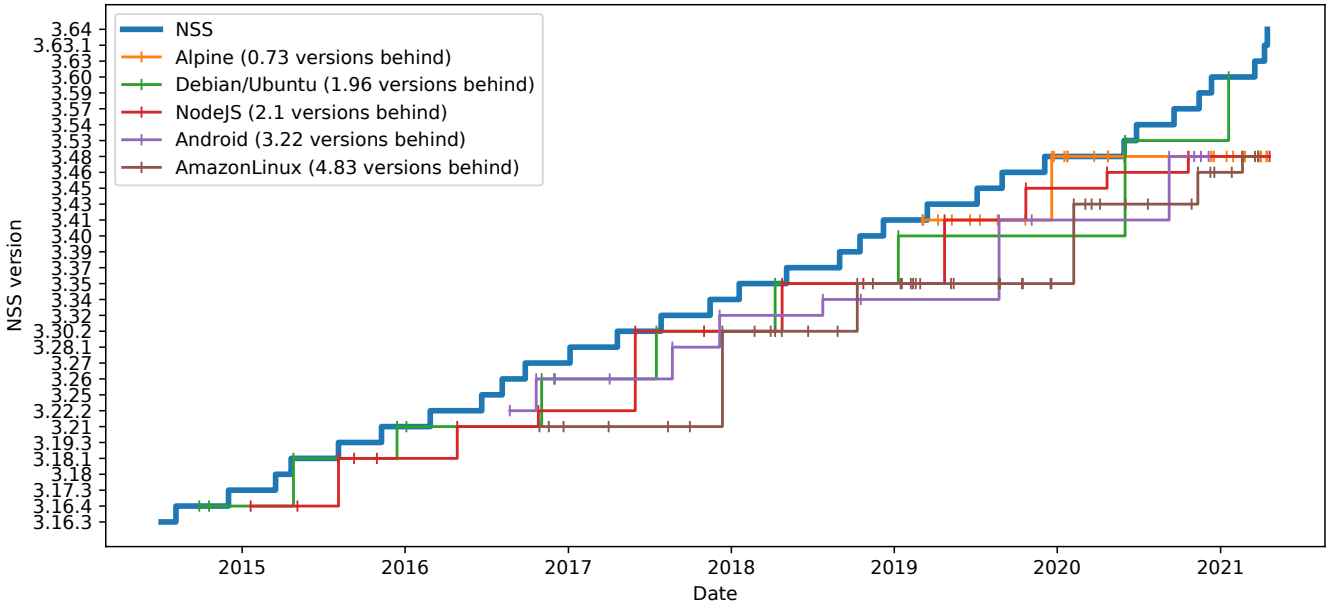

Figure 3: NSS derivative staleness—No derivative root stores match NSS's update regularity. Alpine Linux maintains closest parity to NSS, while AmazonLinux, on average, lags more than four substantial versions behind.

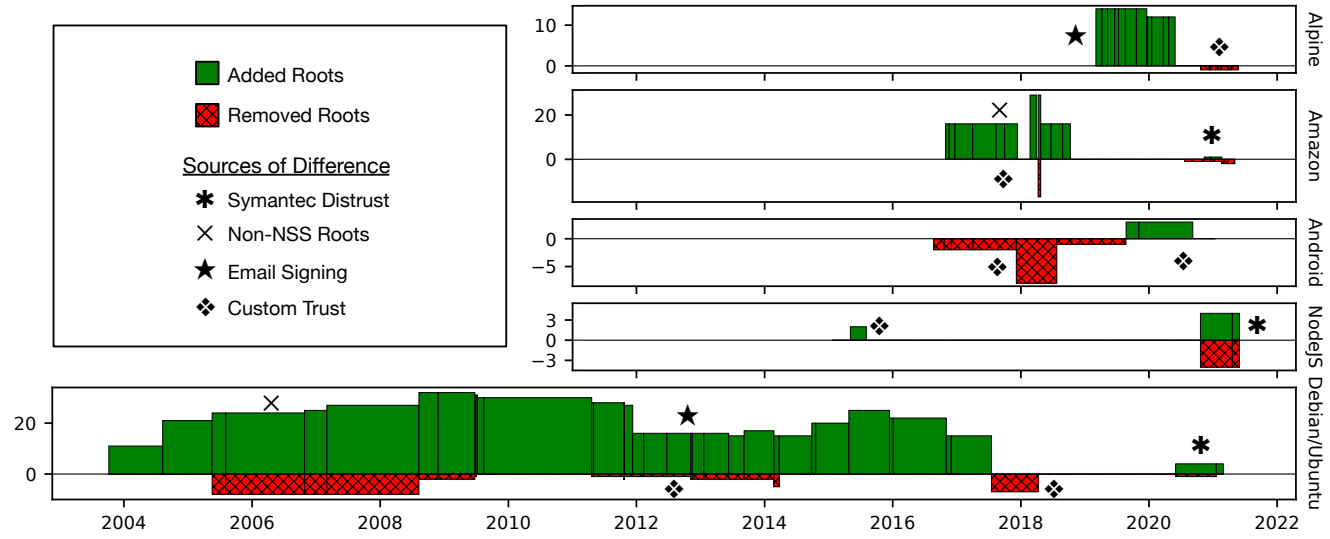

Figure 4: NSS derivative diffs—The number of added/removed root certificates for each NSS derivative indicates that all deviate from strict NSS adherence.

which was included in NSS through other roots, and does not alter the CA makeup of the Amazon Linux root store. Ubuntu/Debian, on the other hand, trusted a total of 19 non-NSS roots, starting from its first snapshot in 2005 and up until 2015. These roots belong to a variety of organizations: the Brazilian National Institute of Information Technology (1), Debian (2), Government of France DCSSI (1), TP Internet Sp. (9), Software in the Public Interest (3), and CAcert(3). Of these CAs, only DCSSI has ever had a root included in NSS. While we cannot trace the inclusion reasons for all these historic roots, we highlight a few interesting cases. The Debian and Software in the Public Interest roots were included to support Debian-specific infrastructure. CAcert, a distributed community CA, was rejected from NSS [35] and other Linux distributions [65] for lack of audits. These practices are a significant departure from NSS inclusion policies and potentially put Ubuntu/Debian users at greater risk.

Email signing One of the fundamental differences between NSS and its derivatives is NSS's ability to specify trust purposes for each root, as well as gradual distrust. Unfortunately, due to their trust store format (single file/directory of root certificates for all purposes), NSS derivatives conflate CAs trusted for TLS with CAs trusted for code signing or email signatures, even though the processes and policies and trust decisions for the three should vary substantially. To quantify a lower bound on the issue, we look at all NSS certificates that have *never* been trusted for TLS and find the derivative root stores that misplace TLS trust in those certificates. We find that Debian/Ubuntu (19 roots) and Alpine (4 roots) all express TLS trust in CA certificates that have never been trusted by NSS for TLS. While Debian/Ubuntu have not trusted such certificates since 2016, Alpine Linux trusted several until 2020. More broadly, we can see that both Debian/Ubuntu in 2017 and Alpine in 2020 shifted from including both TLS server authentication and email signing NSS roots to just include TLS roots.

Customized trust Several derivatives perform customized trust removals. Android never included PSPProcert, which was later removed from NSS, and also preemptively removed the problematic CNNIC root. Both Android and Ubuntu/Debian manually removed WoSign roots, without updating to the latest NSS version which had already removed them. Similarly, Alpine Linux manually removed trust in an expired AddTrust root without updating its NSS version. These instances of manual root store modification in response to CA issues reflect responsible root store management, especially for Android which proactively removes problematic CAs.

Customized trust additions are less easily justified. From 2016–2018, Amazon Linux continually re-added sixteen 1024-bit RSA roots after they had been removed in NSS, and for a brief period in 2018 added thirteen additional expired certificates and CA-requested removals. We could not find a definitive reason for this behavior. NodeJS re-added a deprecated ValiCert root due to OpenSSL chain building issues [44]. These additions are likely necessitated by impact to end users of the derivative root store, which differ from NSS's users and risk calculus.

Takeaways The majority of root providers in our study derive their root stores from NSS, but nearly all derivative root stores demonstrate concerning update and customized trust practices that increase security risk. The root causes for customized trust are a combination of root store design incompatibility, misapplication of NSS roots, proactive removals, and additions to patch downstream bugs. The root causes for update delay are less transparent, but exploring the possible reasons (e.g., operational laxness, rigid deployment timelines, deployment impact to users, etc.) can lead to more efficient and secure NSS copying.

7 DISCUSSION

Even though TLS deployment has blossomed in recent years, the root store ecosystem for TLS sever authentication remains relatively condensed, with essentially three major root programs (Apple, Microsoft, Mozilla) that support a majority of popular user agents. As more devices (i.e., Internet of Things) and applications (e.g., DNS-over-HTTPS) employ TLS in the coming years, existing pain points in the TLS root store ecosystem will become more pronounced. Below, we highlight a few issues and potential solutions.

NSS derivative formats NSS acts as the de facto foundation for root store providers that do not wish to operate their own root store program. Even ignoring update staleness issues (Section 6.1), the derivative root stores in our dataset have struggled to copy NSS with high fidelity due to their inability to indicate partial trust. While it may seem that there is significant inertia behind the simple root certificate file/directory design, since applications expect that interface, Microsoft and Apple already provide TLS interfaces [70, 81] that account for more nuanced root store trust. Given that multitudes more root stores likely already rely on NSS[7], we hope that future work will transition derivatives and new root stores to more modern formats, while maintaining ease of use for developers.

Single purpose root stores Multi-purpose root stores can lead to trust in roots for unintended purposes. As seen with Apple and NSS derivatives, many email signing roots were trusted for TLS server authentication, even though the trusted roots may not have had sufficiently compliant and secure operations for TLS server authentication. More broadly, trust in a root for TLS server authentication does not transfer to other PKI purposes. Multi-purpose root stores can also confuse application developers. In the most egregious case, anecdotal evidence showed that NuGet relied on NSS-copied roots for code signing and timestamping, even though NSS no longer trusts roots for code signing, and has never trusted roots for timestamping. NSS inclusion is a gateway to a wide range of derivative systems that use multi-purpose root stores, and any CA in NSS can issue trusted code-signing certificates in these derivatives without supervision or transparency checks. Moving forward, we recommend a short-term push towards single purpose root stores, such as those recently implemented by RHEL distributions and AmazonLinux (i.e., separate `tls/email/objsign-ca-bundle.pem`). In the long term, we may need a more scalable, cross-platform design for arbitrary trust purposes if the Web PKI expands into an internet-wide permissions system.

Data-informed root trust This study identifies which root programs trust which root CAs but has limited insight into *why* specific CAs are trusted. Anecdotal evidence points to a range of reasons: business relationships between CAs (especially government CAs) and root programs, access to a wider set of subscribers, or simply because a compliant CA requested inclusion. These varied reasons for root inclusion highlight the subjectivity of historical root program policies, which can deviate from the core properties of the Web PKI: scale and security. NSS and other root programs have started increasing objectivity by enforcing the BRs (thereby increasing operational security) and enumerating how new CAs might benefit Mozilla's users (by increasing scale or security) [112]. Prior work such as ZLint [96] is a step towards more objective evaluation, and future work around CA performance and root provider performance is needed. Furthermore, transparency efforts for root program and CA policies/behavior [97, 98] will facilitate the transition towards data-informed root store trust.

[7] Searching `certdata.txt` in Github yields over 200K code results.

8 RELATED WORK

The research community has studied the certificate ecosystem in great depth, with an emphasis on CA behavior and processes. Initial work focused on collecting large certificate datasets [110], evaluating the security of certificate chains [89, 92] (even invalid chains [86]), and issues with the certificate issuance process [83, 84, 96]. More recent work has explored the impacts of CA certificate cross-signing [91] and examined the operational control of CA certificates [98]. This work borrows certificate security metrics used in these prior studies and applies them towards understanding the behavior and processes of root store providers.

Investigations of root store providers are scattered. Some have involved proposals to reduce the large attack surface of root stores, where every root is a single point of failure that can authenticate all domains. Braun et al. performed a user study (n=22) and found that 90% of roots went unused [85]. Smith et al. explored a wide range of root stores and attempted to quantify the minimum set of roots to handle 99% of certificates collected by IPv4 scans [105], but VanderSloot et al. later demonstrated that such scans miss a majority of certificates due to the absence of SNI [110]. Other studies have proposed automatically inferring TLD name constraints on root CAs [93]. Ma et al. helped link CA certificates (roots and intermediates) trusted by Apple, Microsoft, and Mozilla to the CAs that operate them [98]. Our study expands on the set of root stores examined by previous works and focuses on their provenance, CA composition, and security behavior over time.

Several works have examined specific slices of TLS root stores. Vallina-Rodriguez [109] et al. found that 39% of Android root stores included certificates beyond the default Android root store. Device manufacturers and mobile carriers installed a majority of these additional certificates. Korzhitskii et al. identified additional trusted roots in CT logs beyond those used by Microsoft, Apple, and NSS [95], but the security relevance is indirect, since CT log roots are a spam control mechanism. Looking at six network appliances that perform TLS interception, Waked et al. noted that Untangle trusted roots that were immediately vulnerable to MITM attacks [111].

9 CONCLUSION

This work uncovers the inverted pyramid structure of the modern TLS root store ecosystem. A super majority of popular user agents all rely on one of three root programs, which have distinct operational and inclusion practices that reflect differing levels of coverage and risk. TLS root providers have converged on NSS as the basis for nearly all new root stores, causing NSS changes to have an outsized impact on the overall ecosystem. However, NSS copying leads to complacent updates and instances of misuse that we highlight as cautionary examples for future root stores.

10 ACKNOWLEDGEMENTS

We thank our shepherd Oliver Gasser and the anonymous reviewers for their helpful suggestions. We also thank Clint Wilson for insightful discussion. This work is supported in part by a Yunni & Maxine Pao Memorial Fellowship and a gift from DigiCert.

REFERENCES

[1] [n.d.]. About the security partial trust allow list. https://support.apple.com/en-gb/HT204938.
[2] [n.d.]. Add 2 new SECOM root certificates. https://bugzilla.mozilla.org/show_bug.cgi?id=1313982.
[3] [n.d.]. Add Asseco DS / Certum root certificates. https://bugzilla.mozilla.org/show_bug.cgi?id=1598577.
[4] [n.d.]. Add Autoridad de Certificacion Raiz del Estado Venezolano root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1302431.
[5] [n.d.]. Add CA Root certificate (Brazil's National PKI). https://bugzilla.mozilla.org/show_bug.cgi?id=438825.
[6] [n.d.]. Add Chunghwa Telecom's HiPKI Root CA -G1 Certificate to NSS. https://bugzilla.mozilla.org/show_bug.cgi?id=1563417.
[7] [n.d.]. Add Cisco Root CA Cert. https://bugzilla.mozilla.org/show_bug.cgi?id=416842.
[8] [n.d.]. Add D-TRUST Root CA 3 2013 to NSS. https://bugzilla.mozilla.org/show_bug.cgi?id=1348132.
[9] [n.d.]. Add DigiCert non-TLS Intermediate Certs to OneCRL. https://bugzilla.mozilla.org/show_bug.cgi?id=1404501.
[10] [n.d.]. Add Digidentity Service Root Certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1558450.
[11] [n.d.]. Add e-commerce monitoring's GLOBALTRUST 2020 root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1627552.
[12] [n.d.]. Add "Fina Root CA" root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1449941.
[13] [n.d.]. add Finnish Population Register Centre's Root CA Certificates. https://bugzilla.mozilla.org/show_bug.cgi?id=463989.
[14] [n.d.]. Add GLOBALTRUST 2015 root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1440271.
[15] [n.d.]. Add MOI GPKI Root CA certificate(s). https://bugzilla.mozilla.org/show_bug.cgi?id=1226100.
[16] [n.d.]. Add MULTICERT Root Certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1040072.
[17] [n.d.]. Add OATI's Root CA Certificate to Mozilla's trusted root list. https://bugzilla.mozilla.org/show_bug.cgi?id=848766.
[18] [n.d.]. Add PostSignum root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=643398.
[19] [n.d.]. Add PostSignum Root QCA 4 to Root Store. https://bugzilla.mozilla.org/show_bug.cgi?id=1602415.
[20] [n.d.]. Add Renewed AC Camerfirma root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=986854.
[21] [n.d.]. Add Renewed ACEDICOM root certificate(s). https://bugzilla.mozilla.org/show_bug.cgi?id=1239329.
[22] [n.d.]. Add Symantec-brand Class 1 and Class 2 roots. https://bugzilla.mozilla.org/show_bug.cgi?id=833986.
[23] [n.d.]. Add Telia CA root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1664161.
[24] [n.d.]. Add TunRootCA2 root certificate(s). https://bugzilla.mozilla.org/show_bug.cgi?id=1233645.
[25] [n.d.]. Add TunTrust Root CA root certificate. https://bugzilla.mozilla.org/show_bug.cgi?id=1587779.
[26] [n.d.]. Android ca-certificates. https://android.googlesource.com/platform/system/ca-certificates.
[27] [n.d.]. BearSSL. https://bearssl.org/.
[28] [n.d.]. BoringSSL. https://boringssl.googlesource.com/boringssl/.
[29] [n.d.]. Botan: Crypto and TLS for Modern C++. https://github.com/randombit/botan.
[30] [n.d.]. Bouncy Castle. http://git.bouncycastle.org/index.html.
[31] [n.d.]. ca-certificates: Removal of GeoTrust Global CA requires investigation. https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=962596.
[32] [n.d.]. ca-certificates should remove Symantec certs. https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=911289.
[33] [n.d.]. CA/Additional Trust Changes. https://wiki.mozilla.org/CA/Additional_Trust_Changes.
[34] [n.d.]. CA:Camerfirma Issues. https://wiki.mozilla.org/CA:Camerfirma_Issues.
[35] [n.d.]. CAcert root cert inclusion into browser. https://bugzilla.mozilla.org/show_bug.cgi?id=215243.
[36] [n.d.]. CA/Certinomis Issues. https://wiki.mozilla.org/CA/Certinomis_Issues.
[37] [n.d.]. CA/Certinomis Issues. https://wiki.mozilla.org/CA/Certinomis_Issues.
[38] [n.d.]. CA:PROCERT Issues. https://wiki.mozilla.org/CA:PROCERT_Issues.
[39] [n.d.]. CA:Symantec Issues. https://wiki.mozilla.org/CA:Symantec_Issues.
[40] [n.d.]. CA:WoSign Issues. https://wiki.mozilla.org/CA:WoSign_Issues.
[41] [n.d.]. Chrome Root Program. https://www.chromium.org/Home/chromium-security/root-ca-policy.
[42] [n.d.]. CNNIC Action Items. https://bugzilla.mozilla.org/show_bug.cgi?id=1177209.
[43] [n.d.]. cryptlib. https://www.cs.auckland.ac.nz/~pgut001/cryptlib/.

[44] [n.d.]. crypto: add deprecated ValiCert CA for cross cert. https://github.com/nodejs/node/pull/1135.
[45] [n.d.]. Debian ca-certificates. https://salsa.debian.org/debian/ca-certificates.
[46] [n.d.]. Docker hub: alpine. https://hub.docker.com/_/alpine/.
[47] [n.d.]. Docker hub: amazonlinux. https://hub.docker.com/_/amazonlinux.
[48] [n.d.]. Erlang OTP SSL. https://github.com/erlang/otp/tree/master/lib/ssl.
[49] [n.d.]. GnuTLS. https://gitlab.com/gnutls/gnutls/blob/master/README.md.
[50] [n.d.]. Google Groups: dev-security-policy@mozilla.org. https://groups.google.com/a/mozilla.org/g/dev-security-policy.
[51] [n.d.]. Google Groups: mozilla.dev.security.policy. https://groups.google.com/g/mozilla.dev.security.policy.
[52] [n.d.]. Java SE CA Root Certificate Program. https://www.oracle.com/java/technologies/javase/carootcertsprogram.html.
[53] [n.d.]. LibreSSL libtls. https://cvsweb.openbsd.org/src/lib/libtls/.
[54] [n.d.]. MatrixSSL. https://github.com/matrixssl/matrixssl.
[55] [n.d.]. Mbed TLS. https://github.com/ARMmbed/mbedtls.
[56] [n.d.]. Microsec new (ECC) Root Inclusion Request. https://bugzilla.mozilla.org/show_bug.cgi?id=1445364.
[57] [n.d.]. Mozilla CA/FAQ. https://wiki.mozilla.org/CA/FAQ.
[58] [n.d.]. Network Security Services (NSS). https://hg.mozilla.org/projects/nss.
[59] [n.d.]. NodeJS. https://github.com/nodejs/node.
[60] [n.d.]. OkHttp. https://github.com/square/okhttp.
[61] [n.d.]. OpenJDK. http://hg.openjdk.java.net/.
[62] [n.d.]. OpenJDK source. https://github.com/openjdk/.
[63] [n.d.]. OpenSSL. https://github.com/openssl/openssl.
[64] [n.d.]. Removed CA Certificate List. https://ccadb-public.secure.force.com/mozilla/RemovedCACertificateReport.
[65] [n.d.]. Review Request: ca-cacert.org - CAcert.org CA root certificates. https://bugzilla.redhat.com/show_bug.cgi?id=474549.
[66] [n.d.]. Root certificates used by Opera. https://web.archive.org/web/20150207210358/http://www.opera.com/docs/ca/.
[67] [n.d.]. RSA BSAFE. https://community.rsa.com/community/products/bsafe.
[68] [n.d.]. s2n. https://github.com/awslabs/s2n.
[69] [n.d.]. Secure Transport. https://opensource.apple.com/source/Security/.
[70] [n.d.]. Secure Transport. https://developer.apple.com/documentation/security/secure_transport.
[71] [n.d.]. Super-CAs. https://wiki.mozilla.org/CA/Subordinate_CA_Checklist#Super-CAs.
[72] [n.d.]. Symantec root certs - Set CKA_NSS_SERVER_DISTRUST_AFTER. https://bugzilla.mozilla.org/show_bug.cgi?id=1618404.
[73] [n.d.]. Ubuntu ca-certificates. https://launchpad.net/ubuntu/+source/ca-certificates.
[74] [n.d.]. wolfSSL. https://github.com/wolfSSL/wolfssl.
[75] 2005. Apple Root Certificate Program. https://web.archive.org/web/20050503225244/http://www.apple.com/certificateauthority/ca_program.html.
[76] 2010. Windows root certificate program members. https://web.archive.org/web/20110728002957/http://support.microsoft.com/kb/931125.
[77] 2011. Security Update 2011-005. https://support.apple.com/kb/dl1447.
[78] 2015. The MCS Incident and Its Consequences for CNNIC. https://blog.mozilla.org/security/files/2015/04/CNNIC-MCS.pdf.
[79] 2018. Electron's chromium is trusting different CAs then Electron's NodeJS. https://github.com/electron/electron/issues/11741.
[80] 2018. Implement the Symantec distrust plan from Bug 1409257. https://hg.mozilla.org/mozreview/gecko/rev/f6c9341fde050d7079a8934636644aaf54bde922.
[81] 2018. Secure Channel. https://docs.microsoft.com/en-us/windows/win32/secauthn/secure-channel.
[82] Heather Adkins. 2011. An update on attempted man-in-the-middle attacks. https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html.
[83] Bernhard Amann, Robin Sommer, Matthias Vallentin, and Seth Hall. 2013. No attack necessary: The surprising dynamics of SSL trust relationships. In *29th Annual Computer Security Applications Conference*.
[84] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. 2018. Bamboozling Certificate Authorities with BGP. In *27th USENIX Security Symposium (USENIX Security)*.
[85] Johannes Braun and Gregor Rynkowski. 2013. The potential of an individualized set of trusted CAs: Defending against CA failures in the Web PKI. In *International Conference on Social Computing*. IEEE.
[86] Taejoong Chung, Yabing Liu, David Choffnes, Dave Levin, Bruce MacDowell Maggs, Alan Mislove, and Christo Wilson. 2016. Measuring and applying invalid SSL certificates: the silent majority. In *16th ACM Internet Measurement Conference*.
[87] Jeremy Clark and Paul C Van Oorschot. 2013. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *34th IEEE Symposium on Security and Privacy*.
[88] Jon Douglas. [n.d.]. Incident: NuGet Restore Issues on Debian Family Linux Distros. https://github.com/NuGet/Announcements/issues/49.
[89] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. 2013. Analysis of the HTTPS certificate ecosystem. In *13th ACM Internet Measurement Conference*.
[90] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *Network & Distributed System Security Symposium (NDSS '17)*.
[91] Jens Hiller, Johanna Amann, and Oliver Hohlfeld. 2020. The Boon and Bane of Cross-Signing: Shedding Light on a Common Practice in Public Key Infrastructures. In *27th ACM Conference on Computer and Communications Security*.
[92] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL Landscape: A Thorough Analysis of the X.509 PKI Using Active and Passive Measurements. In *11th ACM Internet Measurement Conference*.
[93] James Kasten, Eric Wustrow, and J Alex Halderman. 2013. CAge: Taming certificate authorities by inferring restricted scopes. In *International Conference on Financial Cryptography and Data Security*.
[94] Jeff Kline, Paul Barford, Aaron Cahn, and Joel Sommers. 2017. On the structure and characteristics of user agent string. In *17th Internet Measurement Conference*.
[95] Nikita Korzhitskii and Niklas Carlsson. 2020. Characterizing the Root Landscape of Certificate Transparency Logs. In *IFIP Networking Conference (Networking)*.
[96] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. 2018. Tracking certificate misissuance in the wild. In *39th IEEE Symposium on Security and Privacy*.
[97] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate Transparency. RFC 6962. https://rfc-editor.org/rfc/rfc6962.txt
[98] 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)*.
[99] Mozilla. [n.d.]. Common CA Database. https://www.ccadb.org/.
[100] Mozilla. [n.d.]. WoSign and StartCom. https://docs.google.com/document/d/1C6BlmbeQfn4a9zydVi2UvjBGv6szuSB4sMYUcVrR8vQ/edit.
[101] Johnathan Nightingale. 2011. DigiNotar Removal Follow Up. https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/.
[102] Johnathan Nightingale. 2011. Fraudulent *.google.com Certificate. https://blog.mozilla.org/security/2011/08/29/fraudulent-google-com-certificate/.
[103] Devin O'Brien, Ryan Sleevi, and Andrew Whalley. [n.d.]. Chrome Plan to Distrust Symantec Certificates. https://security.googleblog.com/2017/09/chromes-plan-to-distrust-symantec.html.
[104] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[105] Henning Perl, Sascha Fahl, and Matthew Smith. 2014. You won't be needing these any more: On removing unused certificates from trust stores. In *International Conference on Financial Cryptography and Data Security*.
[106] Ryan Sleevi. [n.d.]. Announcing the Chrome Root Program. https://groups.google.com/g/mozilla.dev.security.policy/c/3Q36J4flnQs/m/VyWFiVwrBQAJ.
[107] Rob Stradling. [n.d.]. authroot.stl. https://github.com/robstradling/authroot.stl.
[108] Wayne Thayer. [n.d.]. DarkMatter Concerns. https://groups.google.com/g/mozilla.dev.security.policy/c/nnLVNfqgz7g/m/TseYqDzaDAAJ.
[109] Narseo Vallina-Rodriguez, Johanna Amann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. 2014. A Tangled Mass: The Android Root Certificate Stores. In *10th ACM Conference on emerging Networking Experiments and Technologies*.
[110] Benjamin VanderSloot, Johanna Amann, Matthew Bernhard, Zakir Durumeric, Michael Bailey, and J Alex Halderman. 2016. Towards a complete view of the certificate ecosystem. In *16th ACM Internet Measurement Conference*.
[111] Louis Waked, Mohammad Mannan, and Amr Youssef. 2018. To intercept or not to intercept: Analyzing TLS interception in network appliances. In *Asia Conference on Computer and Communications Security*.
[112] Ben Wilson. [n.d.]. Quantifying the Value of Adding a New CA. https://groups.google.com/a/mozilla.org/g/dev-security-policy/c/LT_5efOFsSU.
[113] Kathleen Wilson. 2016. https://blog.mozilla.org/security/2016/10/24/distrusting-new-wosign-and-startcom-certificates/.

A POPULAR OS & TLS SOFTWARE ROOT STORES

	Name	Root store?	Details
Operating Systems	Alpine Linux	Yes	Popular Docker image base.
	Amazon Linux	Yes	AWS base image.
	Android	Yes	Most common mobile OS. Also used for Android Automotive.
	ChromeOS	Yes	Proprietary Google product, based on Chromium OS. Excluded because no build target history.
	Debian	Yes	One of the oldest Linux distributions, base of popular OSes such as OpenWRT/Ubuntu.
	iOS / macOS	Yes	Popular mobile / PC Apple devices that use a common root store.
	Microsoft Windows	Yes	Popular PC and server operating system.
	Ubuntu	Yes	Popular desktop Linux distribution, based on Debian.
TLS Libraries	AlamoFire	No	Popular Swift HTTP library.
	Botan [29]	No	Defaults to root store.
	BoringSSL [28]	No	Google fork of OpenSSL used in Chrome/Chromium/Android.
	Bouncy Castle [30]	No	No default, requires configured keystore.
	cryptlib [43]	No	Unknown default.
	GnuTLS [49]	No	Configuration via `–with-default-trust-store-<format>` flag.
	Java Secure Socket Ext. (JSSE) [61]	Yes	`cacerts` JKS file.
	LibreSSL libtls/libssl [53]	No	Configured `TLS_DEFAULT_CA_FILE`.
	MatrixSSL [54]	No	No default, requires configuration.
	Mbed TLS (prev. PolarSSL) [55]	No	No default, requires configuration of `ca_path/ca_file`.
	Network Security Services (NSS) [58]	Yes	Root store in `certdata.txt`, add'l trust elsewhere [33].
	OkHttp [60]	No	Uses platform (e.g., JSSE, BouncyCastle, etc.) TLS.
	OpenSSL [63]	No	Defaults to `$OPENSSLDIR/{certs, cert.pem}`, often symlinked by installer to system certs for Linux. Windows / macOS
	RSA BSAFE [67]	No	Unknown default.
	S2n [68]	No	Defaults to system stores.
	SChannel [81]	No	Defaults to system (Microsoft) store.
	wolfSSL (prev. CyaSSL) [74]	No	No default, requires configuration.
	Erlang/OTP SSL [48]	No	Unknown default.
	BearSSL [27]	No	No default, requires configuration.
	NodeJS [59]	Yes	Static file `src/node_root_certs.h`.
TLS Clients	Safari	No	Uses macOS root store[8].
	Mobile Safari	No	Uses iOS root store.
	Chrome	Yes*	Historically used system roots, with browser control of EV and special cases such as the distrust of Symantec [103]. As of December 2020, the Chrome root store [41] was deployed on ChromeOS and Linux, with full transition for other OSes pending [106].
	Chrome Mobile	No	Uses Android root store.
	Chrome Mobile iOS	No	Uses iOS root store; Apple policy prohibits custom root stores.
	Edge	No	Uses Windows system certificates not via SChannel.
	Internet Explorer	No	Uses Windows system certificates via SChannel.
	Firefox	Yes	Uses NSS root store.
	Opera	No*	Independent root program until 2013 [66]. Uses Chromium (system roots) and Chrome EV.
	Electron	Yes	Chromium + NodeJS application framework that can use roots through both.
	360Browser	Yes	Qihoo browser popular in China. Excluded because no open source history.
	curl	No	Uses libcurl, which can be compiled to use system defaults (e.g., Schannel, SecureTransport) or custom.
	wget	No	Specified in `wgetrc` configuration file. USes GnuTLS, previously OpenSSL.

Table 5: Popular OS & TLS Software Root Stores

B ROOT PROGRAM EXCLUSIVE DIFFERENCES

Cert SHA256	CA	NSS inclusion?	Details
NSS (1)			
beb00b30…	Microsec	Accepted [56]	New elliptic curve root.
Java (0)	–	–	–
Apple (13)			
0ed3ffab…	Gov. of Venezuela	Denied [4]	Microsoft trusts same issuer for email, disallowed on 2020-02 (PSPProcert). Failed NSS inclusion for same issuer due to super CA concerns.
9f974446…	Certipost	–	CA requested cross-sign revocation [9]: cessation of TLS server certs.
e3268f61…	ANF	–	Microsoft trusts same issuer for email, distrust after 2019-02-01.
6639d13c…	Echoworx	–	Microsoft trusted for email.
92d8092e…	Nets.eu	–	Microsoft trusted for email.
9d190b2e…	DigiCert	Accepted [22]	Trusted by Microsoft and NSS for email.
cb627d18…	DigiCert	Accepted [22]	Trusted by Microsoft and NSS for email.
a1a86d04…	D-TRUST	Accepted [8]	Microsoft/NSS trusted for email.
5 roots	*Apple*	–	*Roots for custom Apple Services (e.g., FairPlay, Developer ID)*
Microsoft (30)			
1501f89c…	EDICOM	Denied [21]	Inadequate audits, issuance concerns, CA unresponsiveness.
416b1f9e…	e-monitoring.at	Denied [14]	CA certificate violations of the BRs and RFC 5280.
6e0bff06…	Gov. of Brazil	Denied [5]	Super CA concerns, insufficient auditing / disclosure.
c795ff8f…	Gov. of Tunisia	Denied [24]	Repeated misissuance exposed during public discussion.
407c276b…	Gov. of Korea	Denied [15]	Rejected due to confidential, unrestrained subCAs.
c1d80ce4…	AC Camerfirma	Denied [20]	Numerous issues [34], lead to May 2021 removal of all Camerfirma roots.
ad016f95…	PostSignum	Abandoned [18]	New PostSignum root inclusion attempt running into issues [19].
7a77c6c6…	OATI	Abandoned [17]	No response in 3 years.
604d32d0…	MULTICERT	Abandoned [16]	External subCA concerns and other misissuance. MULTICERT intermediate distrusted in Camerfirma removal [34].
e2809772…	Digidentity	Retracted [10]	
2e44102a…	Gov. of Tunisia	Pending [25]	Community concerns about added-value of the root.
e74fbda5…	SECOM	Pending [2]	Pending since 2016 due to ongoing issue resolution.
24a55c2a…	SECOM	Pending [2]	Pending since 2016 due to ongoing issue resolution.
f015ce3c…	Chunghwa Telecom	Pending [6]	
5ab4fcdb…	Fina	Pending [12]	
242b6974…	Telia	Pending [23]	< 100 leaf certificates in CT.
eb7e05aa…	NETLOCK Kft.		Cross-signed by Microsoft Code Verification Root, which only has kernel-mode code signing permissions.
5b1d9d24…	Gov. of Spain, MTIN	–	Expired in Nov 2019, no intermediates/children in CT.
34ff2a44…	Gov. of Finland	–	Previously abandoned NSS inclusion for a different root [13].
229ccc19…	Cisco	–	< 100 leaf certificates in CT. NSS rejected older root issuing local certificates shipped with Cisco devices [7].
d7ba3f4f…	Halcom D.D.	–	< 100 leaf certificates in CT.
7d2bf348…	Spain Commercial Reg.	–	< 100 leaf certificates in CT.
c2157309…	NISZ	–	< 200 leaf certificates in CT.
608142da…	TrustFactory	–	< 100 leaf certificates in CT.
a3cc6859…	DigiCert	–	WiFi Alliance Passpoint roaming.
68ad5090…	DigiCert	–	Trusted intermediate in NSS/Apple/Java via Baltimore CyberTrust Root.
1a0d2044…	Sectigo	–	Apple/NSS trusted issuer through different root certificate.
3 roots	*Asseco/e-monitoring.at*	*Approved [3, 11]*	*Recently approved by NSS, awaiting addition.*

Table 6: Root Store Differences—Java and NSS rarely implement unique trust, while Apple and Microsoft display more permissive inclusion.

C NSS ROOT REMOVALS

Bugzilla ID	Severity	Removed On	# Certs	Details
1552374	high	2019-07-05	1	Certinomis removal [36]
1392849	high	2017-11-14	3	StarCom removal [113]
1408080	high	2017-11-14	1	PSPProcert removal [38]
1387260	high	2017-11-14	4	WoSign removal [40]
1380868	high	2017-07-27	2	CNNIC removal [78]
682927	high	2011-10-06	1	DigiNotar removal [101]
1670769	medium	2020-12-11	10	Symantec distrust - root certificates ready to be removed
1656077	medium	2020-09-18	1	Taiwan GRCA missisuance: Bugzilla ID: 1463975
1618402	medium	2020-06-26	3	Symantec distrust - root certificates ready to be removed

Table 7: NSS Root Removals—High and medium severity removals from NSS since 2010.