UNDERSTANDING THE TRUST RELATIONSHIPS OF THE WEB PKI

BY

ZANE MA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

ABSTRACT

TLS and the applications it secures (e.g., email, online banking, social media) rely on the web PKI to provide authentication. Without strong authentication guarantees, a capable attacker can impersonate trusted network entities and undermine both data integrity and confidentiality. At its core, the web PKI succeeds as a global authentication system because of the scalability afforded by trust. Instead of requiring every network entity to directly authenticate every other network entity, network entities trust certification authorities (CAs) to perform authentication on their behalf.

Prior work has extensively studied the TLS protocol and CA authentication of network entities (i.e., certificate issuance), but few have examined even the most foundational aspect of trust management and understood which CAs are trusted by which TLS user agents, and why. One major reason for this disparity is the opacity of trust management in two regards: difficult data access and poor specifications. It is relatively easy to acquire and test popular TLS client/server software and issued certificates. On the other hand, tracking trust policies/deployments and evaluating CA operations is less straightforward, but just as important for securing the web PKI.

This dissertation is one of the first attempts to overcome trust management opacity. By observing new measurement perspectives and developing novel fingerprinting techniques, we discover the CAs that operate trust anchors, the default trust anchors that popular TLS user agents rely on, and a general class of injected trust anchors: TLS interceptors. This research not only facilitates new ecosystem visibility, it also provides an empirical grounding for trust management specification and evaluation. Furthermore, our findings point to many instances of questionable, and sometimes broken, security practices such as improperly identified CAs, inadvertent and overly permissive trust, and trivially exploitable injected trust. We argue that most of these issues stem from inadequate transparency, and that explicit mechanisms for linking trust anchors and root stores to their origins would help remedy these problems.

To Dad, Mom, Shawn, and Georgia.

ACKNOWLEDGMENTS

This thesis represents my journey through graduate school, which, thanks to a long list of tremendous people, has brought me much self-growth and joy. I would first like to thank my mom and dad for a lifetime of love and encouragement. It has been a true privilege to have you as parents. I would also like to thank my brother Shawn for being my constant companion and source of inspiration and mindfulness. You beat me at becoming a doctor. . . which I have been happy to abuse by calling you at odd hours for medical advice. Now that I have caught up, my dubious IT tech support is available whenever you need.

I would next like to thank my advisor Michael Bailey for taking a chance on me as your first Ph.D. student at Illinois. You have taught me how to do good research, you've helped me learn a great deal about myself, and you've shown me how to do it all with zest and thoughtfulness. I would also like to thank Zakir Durumeric, for being a mentor and friend who has shown me the research standards that I should constantly strive for. I want to thank Joshua Mason for fostering such a fun, funny, and inviting lab environment. Lastly, I want to thank my thesis committee: Nikita Borisov, Adam Bates, and Gang Wang for their valuable feedback and guidance.

None of my research would have been possible without the stellar researchers that I have been lucky to work with: Amin Kharraz, Andrew Miller, Ariana Mirian, Chad Seaman, Charles Lever, Damian Menscher, David Adrian, Drew Springall, Elie Bursztein, Emily Stark, J. Alex Halderman, Jaime Cochran, Joseph D. Barnes, Joseph Dickinson, Joshua Reynolds, Kaishen Wang, Kurt Thomas, Luca Invernizzi, Manos Antonakakis, Martin Shelton, Matt Bernhard, Meishan Wu, Michalis Kallitsis, Nick Sullivan, Richard Barnes, Robin Berthier, Rohan Subramanian, Siddharth Murali, Taylor Judd, Tim April, Tim Yardley, and Vern Paxson. I would also like to thank the incredible people in the CS department who made the graduate school process easy and delightful, especially Maggie Chappell, Kara MacGregor, and Viveka Kudaligama.

To my labmates through the years, who have become good friends—Deepak Kumar, Paul Murley, Joshua Reynolds, Yi Zhou, Simon Kim, Suyup Kim, Matthew Tischer, Thasphon Chuenchujit, and others—thank you for making the lab special. I really appreciate the countless ways in which you have helped me inside and outside of research, always making sure the servers, and each other, were okay. To my friends, both in Champaign/Urbana and elsewhere, thanks for keeping me sane and ensuring my physical, mental, and emotional wellness. I am fortunate to have such a strong support system, regardless of where I am or what I am doing.

Finally, to my partner Georgia: the warmth, support, and love you have given me these last few years means everything to me. I cannot wait to start this next chapter of life together!

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The web is increasingly adopting Transport Layer Security (TLS) and the web public key infrastructure (PKI). The web PKI is the ecosystem of entities who authenticate, certify, and distribute the public keys that enable TLS, a cryptographic protocol to encrypt client-server connections. Together, TLS and the web PKI secure a myriad of applications, including online banking, social media, email, and any service using HTTPS. Due to such widespread usage, vulnerabilities in either TLS or the web PKI can have severe consequences, such as the trusted forgery of major websites (e.g., DigiNotar compromise [1]) or the potential exposure of millions of private keys (e.g., Heartbleed [2]).

Security researchers and practitioners have made considerable efforts to improve the security of TLS and the web PKI over the last two decades. These endeavors fall into one of three areas: the TLS cryptographic protocol and its implementations, the issuance of authenticated/certified public keys (i.e., certificates), and the distribution of trust anchors (i.e., as a root store) to TLS participants. Despite the precarious trust design of the web PKI—a single compromised trust anchor can forge all identities—prior work has focused primarily on TLS and certificate issuance for three reasons. First, TLS and certificate issuance processes occur broadly across a spectrum of clients and servers. This results in a trove of implementation issues to be fixed. Second, both TLS and certificate issuance are based primarily on protocols rather than policies, making them easier to test and making them more amenable to automated tooling compared to trust management. Third, software and data for trust management is less accessible than TLS and certificate issuance. Most popular TLS client and server software is easily obtained, and developments in internet-wide scanning [3] and Certificate Transparency [4] have greatly increased visibility into the certificate issuance ecosystem. Understanding trust management, on the other hand, often requires undisclosed data about CA operations, root program policies, and real world trust anchor deployments. The dearth of trust management understanding is due to ecosystem opacity.

Judiciously deciding who to trust for authentication is equally as important as securing TLS and certificate issuance, and as the security of TLS and certificate issuance steadily improves, the relative importance of fortifying trust management increases as well. To be more precise, in this dissertation we define trust as the reliance of one party on a separate party to accurately and securely perform a task on the first party's behalf. Trust in trust management appears in several forms: TLS clients relying on CAs for TLS server authentication, root store providers depending on root programs to provide a secure trust anchor store, and TLS user agents relying on root store providers for the same purpose. This dissertation is a novel exploration of two fundamental questions in trust management: who is trusted by whom in the web PKI, and who makes these trust decisions? To answer these questions, we must overcome the opacity challenges of the trust management ecosystem, and in doing so, we reveal latent issues that require improved transparency.

Thesis statement: The absence of explicit trust anchor transparency in the web PKI results in unintended or imprudent trust.

The first component of this dissertation develops novel measurement techniques and datasets to link CA certificates (i.e., the digital representation of trust anchors) with the CAs that actually operate them. This dataset reveals who is trusted when a given CA certificate is included in a root store, and it enables more well-informed trust management decisions and analysis. As an example, our efforts detected an improperly attributed CA certificate that eventually contributed to the removal of the Camerfirma CA from the Mozilla root store.

The second component of this dissertation enumerates and traces the lineage of the root stores trusted by TLS user agents. Our analysis reveals an inverted pyramid of root store trust. A large majority of popular TLS user agents rely on a handful of root store providers that ultimately derive their root stores from one of three foundational sources: Apple, Microsoft, and Mozilla. This trust dependency is not explicitly linked; rather, it is manual and haphazard, resulting in deviations that frequently degrade the security of downstream root stores.

Real world root stores can further deviate from the standard trust anchors provided by operating systems, libraries, and user agents. The third component of this dissertation highlights a general class of exceptions to the default path for trust management: TLS interception. Through a TLS network fingerprinting approach, we find that 4–10% of global HTTPS traffic is intercepted. Additionally, by developing a catalogue of fingerprints, we identify the interception software that operate as a CA and inject their own trust anchor into root stores. Compared to standard CAs that are regularly audited and scrutinized by root store programs, the injected trust anchors are operated by low visibility local CAs embedded in antivirus software or corporate middleboxes. Unsurprisingly, the majority of these interception products expose users to critical security vulnerabilities, and nearly all introduce heightened security risk.

In addition to extending research opportunities, dissolving the first layer of trust management opacity also spotlights the security need for explicit trust anchor transparency. Our work demonstrates that current mechanisms for linking root stores and trust anchors to their origins are either indirect or non-existent, and this opacity yields suboptimal security outcomes. Put another way, we posit that the addition of explicit transparency mechanisms will lead to more judicious and secure trust. For instance, knowledge of CA certificate operators has already influenced root store trust and we believe similar transparency efforts (e.g., connecting root stores to their root program) will positively influence trust management decisions. We ultimately propose new research directions

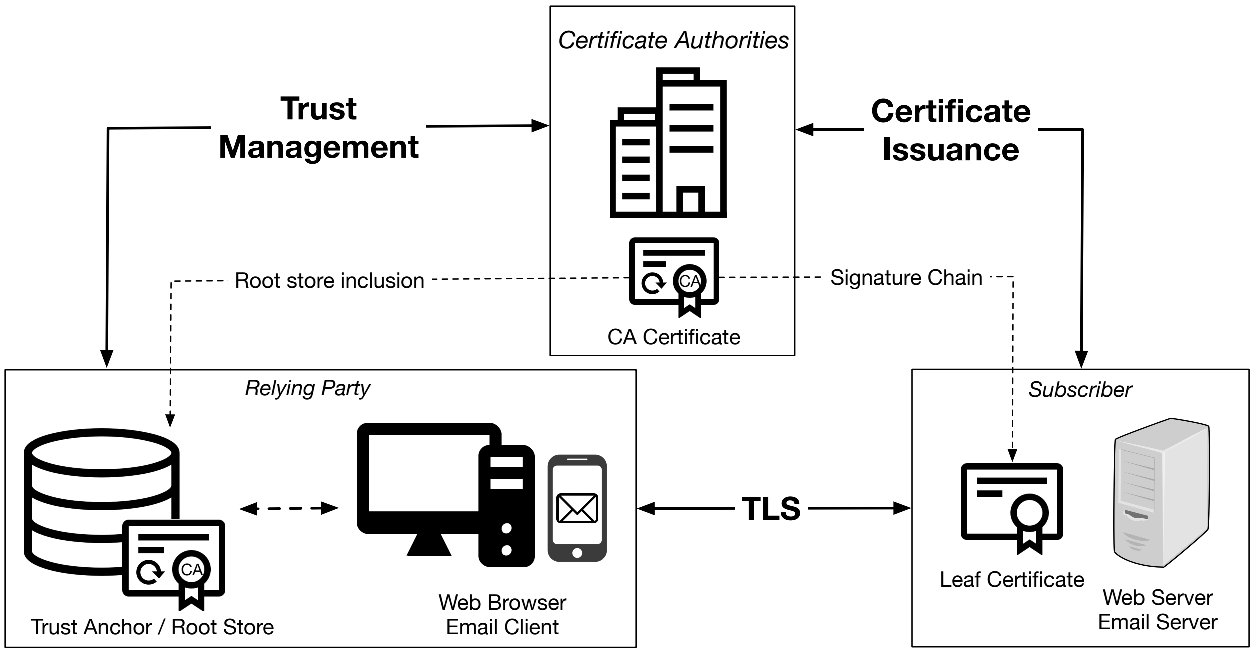


Figure 1.1: **TLS + web PKI overview**—Together, trust management, certificate issuance, and TLS form the delegated authentication system that secures the web.

to kick-start future work in trust management, the web PKI, and delegated authentication systems at large.

1.1 BACKGROUND AND RELATED WORK

This section presents an overview of TLS and the web PKI to better contextualize the contributions of this thesis. We also highlight the extensive body of prior research that has laid the foundation for this work.

TLS and the web PKI comprise a delegated authentication system (Figure 1.1) with three primary participants: subscribers, relying parties, and certification authorities (CAs). The pairwise interactions between these three participants constitute the three broad domains of prior work. Relying parties establish trusted CAs through *trust management*, and subscribers request/receive certificates from CAs via *certificate issuance*. These two domains together constitute the web PKI. Relying parties and subscribers utilize the web PKI as an authentication method for *TLS*, the cryptographic protocol that secures network data end-to-end. We discuss each domain below in more detail.

1.1.1 TLS

TLS, a descendant of the Secure Sockets Layer (SSL) protocol, provides data integrity and confidentiality between a client and server. Data authenticity in most cases (i.e., HTTPS/STARTTLS) is provided externally by the web PKI. Since its standardization in 1999, TLS has evolved through four major versions and continues to be the predominant security protocol on the web. Security research around TLS fits into two general categories: protocol-related work and implementation evaluation. We only present a cursory view of TLS security research and direct readers to [5] for a more comprehensive overview.

A wide range of attacks have targeted TLS/SSL protocol and cryptographic parameters. Some common vulnerable areas are downgrade attacks or insecurity via outdated SSL versions / export ciphers (e.g., Logjam [6], POODLE [7], DROWN [8]); leaky data compression (e.g., CRIME [9], BREACH [10]); and CBC-mode encryption (e.g., BEAST [11], POODLE [7]). Several works have applied formal/symbolic modeling to detect additional attacks [12, 13, 14, 15] and proved formally verified TLS implementations [16, 17, 18, 19, 20].

The security community has discovered many implementation flaws in TLS software. Several works have applied TLS-customized testing techniques (e.g., symbolic execution [21], certificate fuzzing inputs [22], state/blackbox fuzzing [23, 24, 25]) to discover bugs and RFC deviations. The most notable TLS implementation bug, Heartbleed, was discovered by non-academic researchers and revealed promising and problematic community response practices [2]. Beyond TLS for general-purpose web browsing, several studies on the app-ified deployment of TLS [26, 27, 28] have found widespread insecurity in the Android application ecosystem. Others have found issues in alternate TLS applications including payment libraries [29] and email/chat [30]. This thesis adds to the growing list of insecure TLS implementations by testing the security of TLS interception products.

1.1.2 Certificate Issuance

Subscribers that wish to participate in the web PKI (HTTPS servers, for example), request a certificate from a CA that is trusted by the relying parties that the subscriber wishes to communicate with. CAs verify the identity (e.g., DNS name) of a subscriber, and then generate a leaf X.509 certificate that links a subscriber's identity and their public key. To attest to this binding, the CA signs the leaf certificate with its own private key. The corresponding CA public key is represented by a CA certificate. Typically, CAs will sign leaf certificates with an intermediate key for day-to-day use; in turn, this intermediate CA certificate is signed by a root CA key that is offline and more heavily protected.

Initial research into certificate issuance enumerated the ecosystem of CA issuers/subscribers and also evaluated the security of issued certificates. Early work focused on collecting large certificate datasets [31] and evaluating the security of certificate chains [32, 33, 34, 35] (even invalid chains [36]). Recent work has executed BGP attacks on issuance [37] and the impacts of CA certificate cross-signing [38], where CAs sign other CA certificates, thereby creating multiple possible chains. This thesis borrows certificate security metrics used in prior studies and applies them towards understanding the behavior and processes of root store providers.

To combat the issue of insecure and problematic certificates, the CA/Browser (CA/B) Forum established a set of binding *Baseline Requirements* (BRs) in 2011 [39]. The BRs mandate secure as well as hygienic certificate issuance practices (e.g., the subject distinguished name must not have a leading whitespace). Several "linting" tools have been developed to check certificate compliance with the BRs [40, 41]. Although the certificate hygiene BRs do not have direct security consequences, some have suggested that poor certificate hygiene is strongly correlated with instances of certificate insecurity [42, 43]: issuers that don't run a pristine certificate issuance operation are more likely to make security mistakes.

Several defenses have been deployed to protect against certificate misissuance. First, Certificate Transparency (CT) was standardized in 2013, and broadly enforced by 2019 [44], as a reactive defense to force CAs to publicize issued certificates, thereby allowing easy detection of misissuance by the general public [4]. Several works have used CT for other purposes: as a source for discovering phishing DNS names [45, 46, 47, 48], while others have focused on the privacy implications of domain exposure through CT [45, 49]. Second, Certification Authority Authorization (CAA) DNS records were standardized in 2019 as a proactive mechanism to allow domain owners to restrict which CAs may issue leaf certificates for a given domain [50]. Early CAA adoption has been slow, mainly inhibited by CAs and DNS operators [51]. Lastly, in 2019, the IETF standardized Automated Certificate Management Environment (ACME) to automate the certificate issuance process. ACME has contributed to the rapid deployment of HTTPS in recent years, led by Let's Encrypt [52].

So far, academic work has largely focused on the output artifact of issuance, the certificate; no work has examined the software (e.g., EJBCA, Boulder) and configurations responsible for certificate issuance. This thesis takes a first look in this direction by fingerprinting and clustering the issuance profiles for each CA issuer. Attributing these clusters to specific CA software and applying software analysis techniques is a promising avenue of future work.

1.1.3 Trust Management

Relying parties are any entity, such as a TLS client, that relies on a valid certificate to bind an identity (e.g., DNS name) to a public key. Prior to performing TLS, a relying party establishes a set of trust anchors, also known as a root store, that represent CAs that the relying party trusts to authenticate subscribers. Trust anchors are often stored as an X.509 certificate [53], which contains a CA's name and public key. Not every relying party handpicks their own trust anchors—as demonstrated later in this thesis, many TLS clients defer to the root store provided by the underlying operating system or other root store providers.

The trust management ecosystem is opaque and poorly studied. A few scattered works have examined specific slices of TLS root stores. In 2014, 39% of Android root stores were found to include certificates beyond the default Android root store, primarily installed by device manufacturers and mobile carriers [54]. Another study identified additional trusted roots in CT logs beyond those used by Microsoft, Apple, and NSS [55], but relevance to TLS is indirect, since CT log root stores are a spam control mechanism and not a security feature. Lastly, one study examined six network appliances that perform TLS interception and noted that one product Untangle relied on insecure roots that were immediately vulnerable to MITM attacks [56].

The brittle, fault-intolerant design of web PKI trust has prompted many to propose design modifications and replacement systems. Several works attempt to reduce the large attack surface of root stores, where every root is a single point of failure that can authenticate all domains. A user study (n=22) in 2013 found that 90% of roots went unused [57], and subsequent work in in 2014 explored several root stores and attempted to quantify the minimum set of roots to handle 99% of certificates collected by IPv4 scans [58]. Unfortunately, VanderSloot et al. later demonstrated that such scans miss a majority of certificates due to the absence of SNI [31]. Other studies have proposed automatically inferring TLD name constraints on root CAs [59]. Some have proposed systems that forgo delegated authentication entirely [60]. Unfortunately, none of these proposals have gained adoption, largely due to the entrenchment of the web PKI's global deployment, and recent work has instead explored a system to transition from today's web PKI to arbitrary alternatives [61].

This thesis addresses the opacity of trust management by taking a broad look at who is trust by whom in the web PKI. We develop new techniques and datasets to explore which CAs operate which CA certificates, which CA certificates are included in which root stores, and which CAs are injected into default root stores. Our characterization of the trust management ecosystem provides a new perspective that can inform the development of design modifications and replacement systems.

1.2 CONTRIBUTIONS

The main contributions of this dissertation are:

- **New fingerprinting methods to detect previously opaque aspects of the web PKI at scale.** We developed two fingerprinting techniques to shed light on two blackbox processes: certificate issuance by CA software, and TLS interception by middleboxes or end-host software.

  - *X.509 certificate ASN.1 fingerprinting [62].* ASN.1 is a tree-like type-length-value (TLV) data format. By fingerprinting the ordered structure of a X.509 certificate's ASN.1 tree rather than the terminal node values, we were able to distinguish data entropy arising from different certificate issuance systems and configurations. Each CA issuer's certificate profile (collection of fingerprints for issued certificates) provides insight into shared CA software and issuance operations.
  - *TLS client fingerprinting [63].* TLS handshakes specify a preconfigured list of cryptographic capabilities that can identify different TLS client software and configurations. We construct a catalogue of TLS client fingerprints, including heuristics for software with dynamic fingerprints, and compare them with HTTP User-Agent headers in HTTPS connection; mismatch between the two indicates likely TLS interception.

- **Identification of CA certificate operators.** We developed a new tool `Fides` that combines three CA operational perspectives—network infrastructure, issuance software/configuration, and audits—along with the Common CA Database (CCADB) to identify which CAs operate which CA certificates. The novelty of this analysis reflects its primary challenge, the lack of ground truth data, which we address through a manually generated evaluation dataset and output consistency checks. We deploy this system over all 2.9B certificates present in CT and produce a new dataset of CA operations. This new perspective reframes prior research on the certificate issuance ecosystem (e.g., the web PKI contains 75% fewer CAs than suggested by prior studies) and facilitates future trust management decision making. We open source our dataset of 6.8K CA certificates, their operational fingerprints, and CA operator labels [64].

- **Characterizing the (in)security of TLS interception products.** Applying TLS client fingerprints and HTTP User-Agent analysis to 7.75B HTTPS connections indicates that 4–10% of global HTTPS traffic is intercepted, a tenfold increase over prior detection efforts. To understand the security impact this interception, we test over twenty interception products on a variety of systems and browsers to assess the soundness of their certificate validation, the strength of their cryptographic parameters, and vulnerability to known attacks. Nearly all interception products increase user exposure to security risks, especially middlebox products (58% severe vulnerabilities) that are typically utilized in corporate networks. Outside of the US, mobile provider networks exhibit above average interception rates. Within the US, residential/business networks are the most commonly intercepted.

- **Discovering root store provenance in the web PKI.** Grounded in the top 200 user agents observed by a global CDN, we collect and trace the root store dependencies of eight major operating systems, twenty TLS libraries, and thirteen TLS web clients. This dataset and subsequent provenance clustering reveals the inverted pyramid structure of the modern TLS root store ecosystem, based on three root programs: Apple, Microsoft, and Mozilla. Of these, NSS displays relatively transparent, restrictive, and agile root store management. Perhaps unsurprisingly, all derivative root stores copy from NSS. This sturdy foundation does not, however, imply strong root store security; all NSS-derivatives deviate from their upstream root store due to staleness, root store misuse, or bespoke trust decisions that frequently degrade security.

1.3 DISSERTATION ROADMAP

The remainder of this dissertation is structured as follows. The next chapter, Chapter 2, takes a top-down view of trust management and identifies the CAs who operate the CA certificates that are included in root stores. We then explore the root programs that choose default CAs to trust in Chapter 3 and determine which TLS user agents are which root stores. Next, we detect a major class of deviations to default root store trust, TLS interceptors, and evaluate their security impact in Chapter 4. Finally, Chapter 5 presents the broader insights of this work and offers future research directions in further extending web PKI transparency and developing a principled systems framework for generalized PKI design.

CHAPTER 2: CA CERTIFICATE OPERATORS

The first step to trust management transparency is to identify the entities involved. Although root stores directly store trust anchors (e.g., X.509 certificates), these certificates are proxies for the CAs that operate the keys indicated in the certificate. We first draw attention to the disconnect between CA certificate names and the CAs that actually operate them. No definitive source of this information exists, so we then describe a new methodology to measure CA operational features and then link CA certificates with evidence of shared operations. We then validate our methodology and release a new dataset of 6K CA certificates and their likely CA operator. This information is essential for informed trust decisions. For example, in the wake of Symantec's distrust by major browsers, it was imperative to identify all CA certificates operated by Symantec. Most CA certificates contained names that weakly linked them to Symantec, and a single omission would have left the entire browser ecosystem at continued risk of Symantec malfeasance.

2.1 BACKGROUND AND MOTIVATION

TLS depends on a supporting Public Key Infrastructure (PKI), which provides a scalable mechanism for mapping network identifiers (e.g., domain names) to cryptographic keys. In this section, we outline the key parties in the Web PKI and their roles (Figure 2.1). We refer the reader to [5] for an in-depth introduction to the Web PKI.

2.1.1 Certificate Identity and Control

Certificates link an identity to a public key. For subscriber certificates, this identity is the domain or IP address that was validated during the issuance process. However, for CA certificates, the identity is the name of the organization. Specifically, the certificate Subject Common Name (CN) can be any unique string to help the CA identify the certificate, but the Subject Organization must contain the Subject CA's name or a doing-business-as (DBA) / fictitious business name [39].

CA certificates often live longer than CAs themselves, and a certificate's subject can be misleading in the case of a merger or acquisition, or if a CA decides to sell a root to another company. For example, as can be seen in Figure 2.2, Symantec/DigiCert and Comodo/Sectigo control two certificates that both appear to belong to UserTrust. UserTrust was an independent CA that transferred several of its root certificates to GeoTrust [65], which was acquired by VeriSign [66], then Symantec [67], and ultimately DigiCert [68]. UserTrust and its remaining root certificates were acquired by Comodo [69], which eventually rebranded as Sectigo [70]. While in some cases, it

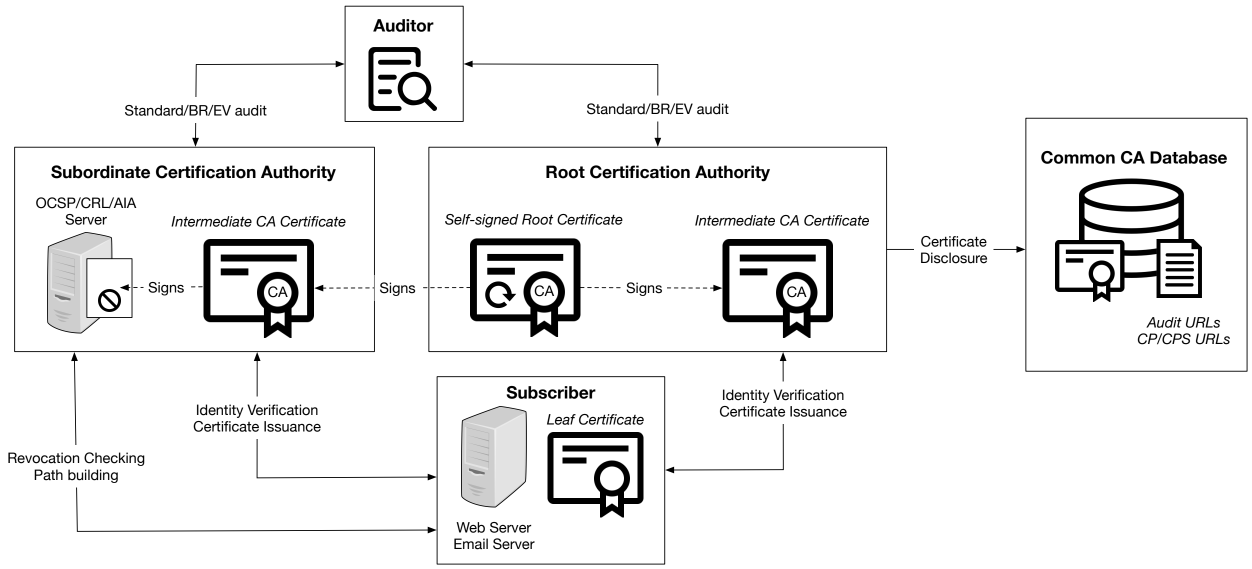


Figure 2.1: **CA PKI overview**—Root CAs can issue intermediate CA certificates for their own use or for independent subordinate CAs, which operate separate issuance and revocation/path building infrastructure. CAs are required by the NSS root store to disclose audit and policy document URLs for CA certificates through CCADB.

Symantec / DigiCert operated root c38dcb389593…

```
commonName          = UTN-USERFirst-NetworkApplications
orgUnitName         = http://www.usertrust.com
orgName             = The USERTRUST Network
localityName        = Salt Lake City
stateOrProvinceName = UT
countryName         = US
```

Comodo / Sectigo operated root 43f257412d44…

```
commonName          = UTN-USERFirst-Client Authentication and Email
orgUnitName         = http://www.usertrust.com
orgName             = The USERTRUST Network
localityName        = Salt Lake City
stateOrProvinceName = UT
countryName         = US
```

Figure 2.2: **Misleading Names**—The Subject fields of two roots previously operated by Symantec/DigiCert and Comodo/Sectigo illustrate that 1) the names in CA certificates do not reflect their operators, and 2) similar certificate names have no bearing on shared control.

is possible to reassemble a CA certificate's history, many business transactions occur in private and there is often no paper trail that explicitly lays out the transfer of ownership/control of a CA certificate.

In most cases, we are most interested in who controls a CA certificate—the entity that has operational access to the cryptographic keys associated with a certificate and is responsible for the certificates issued by those keys. In this work, we consider a CA certificate operator to be the legal entity that controls the hardware security module (HSM) containing a CA certificate's private key. Intermediate certificates (if technically unconstrained) inherit the trust given to root certificates, but they don't necessarily inherit the same operator. Intermediate certificate control falls into three categories:

1. The intermediate is controlled by the root CA. This is common practice for all root CAs that wish to issue leaf certificates.

2. The intermediate is controlled by the root CA, but legally belongs to a subordinate CA. For example, Sectigo runs a white-labeled CA service for Web.com/Network Solutions [71], but Network Solutions owns the intermediate certificates issued by Sectigo [72].

3. The intermediate is controlled and owned by a subordinate CA. This often occurs when a new CA, such as Let's Encrypt, wants to bootstrap trust through an existing root CA [73].

This work focuses on understanding operational control, distinguishing scenarios 1–2 from scenario 3, rather than legal ownership, which requires more legal expertise.

2.1.2 User Agent Root Stores

Every User Agent (e.g., web browsers) that validates certificates ships a set of trusted "root" CA certificates that serve as the root of trust in the Web PKI. CAs rarely use root certificates to directly sign leaf "subscriber" certificates (e.g., the certificate for a website). Rather, root certificates sign intermediate CA certificates, which handle day-to-day signing of subscriber certificates. Root certificates can thus remain offline, protecting them from compromise. This is a necessary precaution due to the difficulty of updating root stores. While new roots are added/removed as CAs emerge, dissolve, or adopt new technology, it can take years for a new root to propagate to clients and become globally reliable. Intermediate CA certificates are also used to delegate trust to third parties.

Products typically have their own root stores or borrow the root store of another product; each product also has its own requirements for including a CA in its root store. For example,

	Organization	#	Symantec Affiliation
Blacklisted Roots	Symantec	10	–
	VeriSign	14	Acquired by Symantec (2010) [67]
	TC TrustCenter	10	Acquired by Symantec (2010) [75]
	GeoTrust	8	Acquired by VeriSign (2006) [66]
	Equifax	4	Acquired by GeoTrust (2001) [76]
	UserTrust	1	GeoTrust partnership (2001) [65]
	Thawte	10	Acquired by VeriSign (1999) [77]
	RSA Data Sec.	1	Spun out VeriSign (1995) [78]
Whitelisted	Apple	6	Sub-CA intermediates
	Google	1	Sub-CA intermediates
	DigiCert	2	Cross-signed DigiCert roots
	DigiCert	2	Transition intermediates

Table 2.1: **Symantec Distrust**—Blacklisting of Symantec-controlled roots involved 58 root certificates [79] with 8 separate orgs. in their X.509 Subject field. These orgs. are linked through a scattered history of corporate spin-offs and acquisitions.

Mozilla requires roots to publicly disclose unconstrained intermediates in Common CA Database (CCADB) [74], a Mozilla-operated repository, while Microsoft does not. CAs demonstrate their compliance with root store requirements by publishing Certificate Policies (CP) and Certification Practice Statements (CPS) that describe how the CA operates. CAs then enlist a third-party accredited/licensed auditor to verify the CA's compliance with their own written policies and public standards like the CA/B Forum Baseline Requirements [39], and either the European Telecommunications Standards Institute (ETSI) criteria or the WebTrust criteria. The CP, CPS, and audit documents provide a detailed look at a CA's operations and management. Several browser operators, including Microsoft and Mozilla, require that root CAs register their CA certificates along with links to audit/CP/CPS documentation in CCADB.

2.1.3 Operational Consequences

CA ownership data is critical to root store operators as exemplified by the distrust of Symantec roots in 2017. Between 2009–2017, Symantec repeatedly misissued certificates [80], and as a result, Google Chrome [81], Mozilla [82], Apple [83], and Microsoft [84], discontinued their trust in Symantec-issued certificates. Identifying Symantec-controlled CA certificates required significant manual investigation of CA audits, CA operational characteristics, and corporate ownership structure. In total, Chrome blacklisted 58 root certificates belonging to eight Subject Organizations, seven of which had no direct indication of Symantec ownership (Table 2.1). Intermediate CA certificates also require attribution because root CAs can delegate intermediate certificates to independent "subordinate" organizations. For example, even after all Symantec roots were distrusted, not all of their child intermediates were subject to distrust. Apple and Google both operated subordinate-CAs (sub-CAs) that chained to Symantec's roots, and these intermediates were explicitly whitelisted and exempt from distrust due to their independent operation.

2.1.4 Research Consequences

Since no methods have previously existed for mapping CA certificates to their operators, existing research has defaulted to the information available in X.509 chains when attempting to characterize the CA ecosystem. For example:

CA ecosystem. Studies of the CA and certificate ecosystem [31, 33, 36, 42, 85] have aggregated certificates based on their Subject Organization names. Figure 2.3 provides a brief comparison between the perspectives provided by Subject Org. names and CCADB, which maps more closely to CA operation as detailed in the following section. At the root certificate level, Subject Orgs. and CCADB labels are within the same order of magnitude. However, the number of Subject Orgs. for intermediate certificates significantly exaggerate the diversity of the CA ecosystem. Similarly, the number of CAs responsible for 50% of the CA ecosystem shrinks from 54, based on Subject Org., to 4, based on CCADB data. Reframing prior studies and performing future studies under the context of CA certificate control will lead to more accurate and actionable results.

BGP attacks on domain validation. Previous work in 2018 [37] identified that Symantec was vulnerable to BGP hijacking attacks during domain control verification. However, in 2017 DigiCert acquired Symantec's Website Security and PKI solutions, and Symantec's CA certificates had transitioned to DigiCert operational control by December 1, 2017 [86]. Future analysis could identify whether the reported issues were specific to Symantec and its web of CA acquisitions (Section 2.1.3), or if they were systemic throughout DigiCert, which is the largest CA by distribution of roots and intermediates.

Phishing certificates. A study from 2019 [48] identified the top ten issuers of phishing certificates, listing Let's Encrypt as the most common issuer at 34.4%, followed by cPanel at 22.2%, and RapidSSL at 9.1%. When we take CA operators into account, we find that cPanel is actually operated by Sectigo, and that Sectigo controls three (cPanel, COMODO RSA, COMODO ECC) of the top five most common issuers. Similarly, DigiCert operates four (RapidSSL TLS RSA, CloudFlare Inc ECC CA-2, DigiCert SHA2, RapidSSL CA) of the top ten phishing certificate issuers. Taken together, the top three CAs issuing the most phishing certificates would be Let's

	Certs	Subject orgs	CCADB owners
All Trusted Roots	366	178	130
Microsoft Roots	354	176	130
Apple Roots	166	83	60
NSS Roots	147	72	52
All Trusted Intermediates	3,447	637	90
All Trusted Certs	3,813	685	132

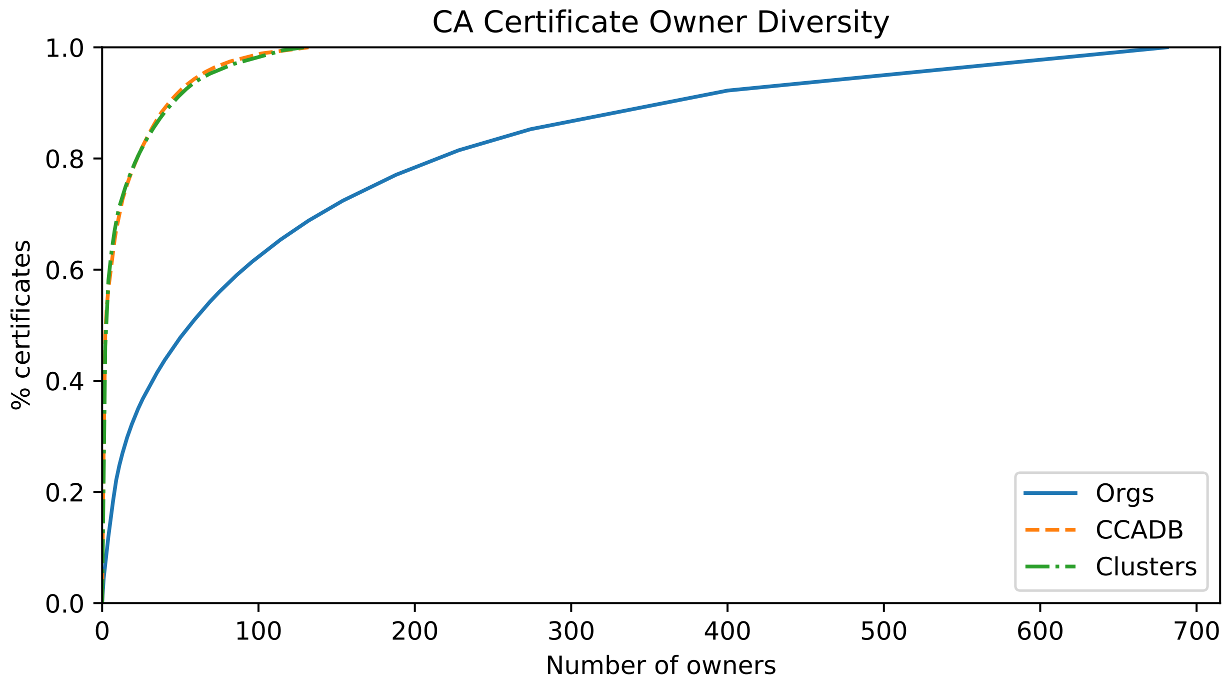


Figure 2.3: **CA Certificate Owner Perspectives**—Certificate Subject organization names exaggerate the size of the CA ecosystem. CCADB hints at more condensed CA certificate control, with a few major players.

Encrypt (34.4%), Sectigo (32.8%), and DigiCert (18.9%). This CA control perspective reveals that phishing certificates are concentrated not just within Let's Encrypt, but Sectigo as well.

2.1.5 Potential Sources of Truth

CCADB—the CA certificate database that major browsers jointly maintain—presents an enticing alternative to the unreliable Subject fields found in CA certificates. Although CCADB was publicly accessible as early as 2016, academic PKI research has largely overlooked it. CCADB's low utilization likely stems from poor public awareness and its misalignment with CA operations.

CCADB does not currently have a field for the legal entity that manages each intermediate or root certificate. The closest field is "CCADB owner" field, which denotes the Salesforce account that is responsible for administrative reporting. For example, although DigiCert acquired QuoVadis and assumed control of all CA operations in January 2019 [87], both DigiCert and QuoVadis exist as separate CCADB owners as of July 2020. Historically, independent subordinate CAs were also disclosed under their root CA in CCADB (e.g. Apple intermediates were disclosed under DigiCert and Sectigo). To address this discrepancy, in March 2019 CCADB began reporting intermediate CA certificates with their own audit statements that are not inherited from the parent certificate. Since independent audits often indicate independent operation, this reporting expanded the utility of CCADB for mapping CA certificate control. However, even CCADB's subordinate CA labels are not necessarily representative of actual CA certificate control. For example, as detailed in Section 2.3.1, Let's Encrypt's cross-signed certificates from IdenTrust are not disclosed as an independent subordinate CA, since they are listed under an IdenTrust audit (which states, ironically, that the cross-signs are not covered by the audit).

Second, not all CA certificates are disclosed through CCADB, since not all root store operators (e.g., Apple) require public disclosure. Furthermore, Mozilla only requires CCADB disclosure for technically unconstrained certificates, which allows for certificates to go unlabeled. Subsequent analysis in Table 2.2 reveals that 665 (20%) of trusted issuers (by Subject + SPKI) are missing from CCADB.

Audit, CP, and CPS documents supplied by CAs initially appear to provide the basis for identifying the organization that controls each CA certificate. However, this is often not possible in practice. For example, in the case of the Symantec distrust, Thawte, GeoTrust, and Symantec all submitted independent audits that did not indicate the relationship between the companies. Furthermore, audits are composed of plaintext English rather than structured data, which would require manual analysis for thousands of CA certificates to uncover ownership details.

	Issuers	**CCADB**	**Cert FPs**	**Cert URLs**	**Audits**	`Fides`
All Trusted Roots	*359*	*354 (98.6%)*	*330 (91.9%)*	*296 (82.5%)*	*204 (56.8%)*	*343 (95.5%)*
Microsoft Roots	352	352 (100.0%)	325 (92.3%)	292 (83.0%)	203 (57.7%)	337 (95.7%)
Apple Roots	165	158 (95.8%)	164 (99.4%)	157 (95.2%)	123 (74.5%)	164 (99.4%)
NSS Roots	147	147 (100.0%)	144 (98.0%)	143 (97.3%)	128 (87.1%)	147 (100.0%)
All Trusted Intermediates	*3,058*	*2,375 (77.7%)*	*1,858 (60.8%)*	*1,783 (58.3%)*	*1,736 (56.8%)*	*2,583 (84.5%)*
Microsoft Intermediates	3,031	2,364 (78.0%)	1,844 (60.8%)	1,773 (58.5%)	1,725 (56.9%)	2,558 (84.4%)
Apple Intermediates	2,493	2,168 (87.0%)	1,502 (60.2%)	1,469 (58.9%)	1,522 (61.1%)	2,110 (84.6%)
NSS Intermediates	2,366	2,135 (90.2%)	1,381 (58.4%)	1,355 (57.3%)	1,564 (66.1%)	2,019 (85.3%)
All Trusted Certs	*3,338*	*2,673 (80.1%)*	*2,111 (63.2%)*	*2,007 (60.1%)*	*1,909 (57.2%)*	*2,847 (85.3%)*
All CA Certs	*6,549*	*4,845 (74.0%)*	*4,363 (66.6%)*	*3,898 (59.5%)*	*2,868 (43.8%)*	*5,613 (85.7%)*

Table 2.2: **Data Coverage**—`Fides`'s combined datasets miss sixteen trusted root issuers (subject+SPKI), and include 84.5% of trusted intermediate CA issuers.

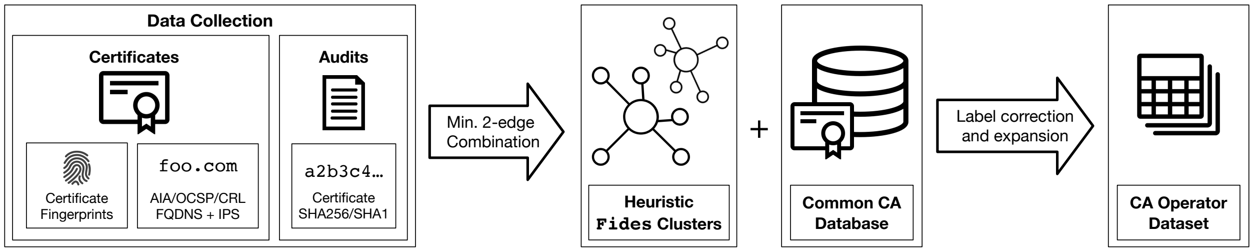


Figure 2.4: `Fides`—Integration of certificate- and audit-based data sources creates heuristic clusters that approximate shared CA certificate control. Combined with CCADB, `Fides` outputs a dataset of CA certificates and their operators.

2.2 `FIDES`: A SYSTEM FOR UNCOVERING OWNERSHIP

Building on the observation that certificates issued by the same organization are likely to be structured similarly, we introduce `Fides`, which we use to aggregate and label CA behavior through fingerprinting of certificate generation software, network infrastructure, and audit details. By clustering on these features, we can detect when CA certificates are controlled by the same party, and using CCADB to seed our labeling process, we build a new dataset that more accurately depicts CA certificate control.

2.2.1 Data Collection

We began our investigation by collecting 2.9B certificates available prior to July 1, 2020 from all CT logs trusted by Google Chrome or Apple [88, 89]. We observed 121,482 unique CA certificates and then filtered out 117,106 ad hoc CA certificates issued by Google to check CT server uptime [90]. We also included 2,240 CA certificates that were disclosed in CCADB, but not present in CT. We labeled the CA certificate data with the trusted root certificates for Apple, Microsoft, and Mozilla NSS as of July 1, 2020 as well as revocation data from NSS OneCRL and Chrome CRLSets revocation lists.

Although CT provides a complete certificate chain for a given certificate (that leads to a trusted root for a given CT log), the presented certificate chain may represent just one of many valid certificate chains. For example, consider a chain of two certificates, *A* and *B*, where *A* is the issuer of *B*. Now consider a third certificate *C*, that has the same Subject+Subject Public Key Info (SPKI) as *A*. The chain of *A*–*B* does not preclude the possibility that *C* actually issued *B*, since *C*–*B* is a valid chain as well. This is a consequence of the flexible design of X.509 certificate chaining, which considers any certificates with the same Subject+SPKI (SSPKI) to be interchangeable parents. Given a child certificate, only the SSPKI of the parent certificate(s) can be determined, and we perform parent/child analysis at the granularity of unique SSPKI pairs.

```
16 # x509 certificate root
   16 # TBS certificate
      0
         2
      2
      16 # Signature Algorithm
         6.1.2.840.113549.1.1.11 # sha256WithRSAEnc.
         5
      16 # Validity
         23
         23
      16 # Subject
         17
            16
               6.2.5.4.3 # Common Name
               19
      16 # Subject Public Key Info
         16
            6.1.2.840.113549.1.1.1 #rsaEncryption
            5
         3
      3 # Extensions
         16
            16
               6.2.5.29.15 # Key Usage
               1 # Critical
               4
                  3
                     100001 # digSig, keyCertSign
            16
               6.1.3.6.1.5.5.7.1.1 # AIA
               4
                  16
                     16
                        6.1.3.6.1.5.5.7.48.1 # OCSP
                        6
                     16
                        6.1.3.6.1.5.5.7.48.2 # Iss.
                        6
   16
      6.1.2.840.113549.1.1.11 # sha256WithRSAEnc.
      5
   3
```

Figure 2.5: **Sample certificate fingerprint**—Each node label is the ASN.1 universal tag type [91], with OID values added as a suffix for OID tag type 6. Extensions may override default ASN.1 tag types.

As part of our data acquisition, we independently verified the certificate chains for each CA certificate and discovered 32 certificates containing signature algorithm inconsistencies[1]. We removed these certificates, yielding a final dataset of 2.9B trusted leaf certificates and 9,154 CA certificates accounting for 6,549 unique SSPKI pairs (Table 2.2).

2.2.2 Fingerprinting Leaf Certificates

We first analyze the ASN.1 structure of the leaf certificates that each CA certificate has signed and then identify clusters of consistent certificate structure. This is possible because CAs have

[1]The signatureAlgorithm field in the TBS Certificate does not match the second signatureAlgorithm field after the TBS Certificate, or does not contain a known OID.

considerable freedom in how the certificates they generate are structured, particularly for the fields in the Subject DN and data included in X.509 extensions. For example, one CA's certificate generation software might only create certificates with 2048-bit RSA public keys, while another may always include both HTTP- and LDAP-based revocation URLs. X.509 certificates are structured in an ordered tree, following the hierarchical ASN.1 data format. Each non-leaf ASN.1 node, including the root, represents a compound ASN.1 field that has one or more sub-fields. Each leaf node contains the value for a field, which can be a string, integer, OID, etc.

We analyze a certificate's ASN.1 tree structure without leaf node values as our certificate fingerprint abstraction. This is because while certificate structure is relatively stable, the values within the structure are not. For example, a certificate's public key should be generated at random. Excluding leaf node values from an ASN.1 certificate focuses on differences caused by different certificate generation software/configuration, rather than differences arising from required high-entropy fields (e.g., serial number) or user input (e.g., Subject Name). The one general exception to this is enumerable values that are denoted by an Object Identifier (OID) ASN.1 node, which do not introduce high entropy. Extension types, for instance, are specified by an OID and are more indicative of software configuration rather than input diversity or required high-entropy fields.

Figure 2.5 provides a sample fingerprint, which demonstrates some of the certificate properties that it captures: the type of cryptographic keys as well as the type and order of extensions. Issuer Names are excluded from the fingerprint, since the absence or presence of the components are dictated by specific issuing certificates rather than certificate generation software. For extensions, X.509 certificates abstract extension data into an ASN.1 *octet string* field so that new and unknown extensions can be safely parsed. Extensions often have custom data formats that override the standard ASN.1 types, so we implement custom parsers for extensions to further increase the precision of our fingerprints. To encourage further research with this technique, we have open sourced our certificate ASN.1 fingerprinting tool [62].

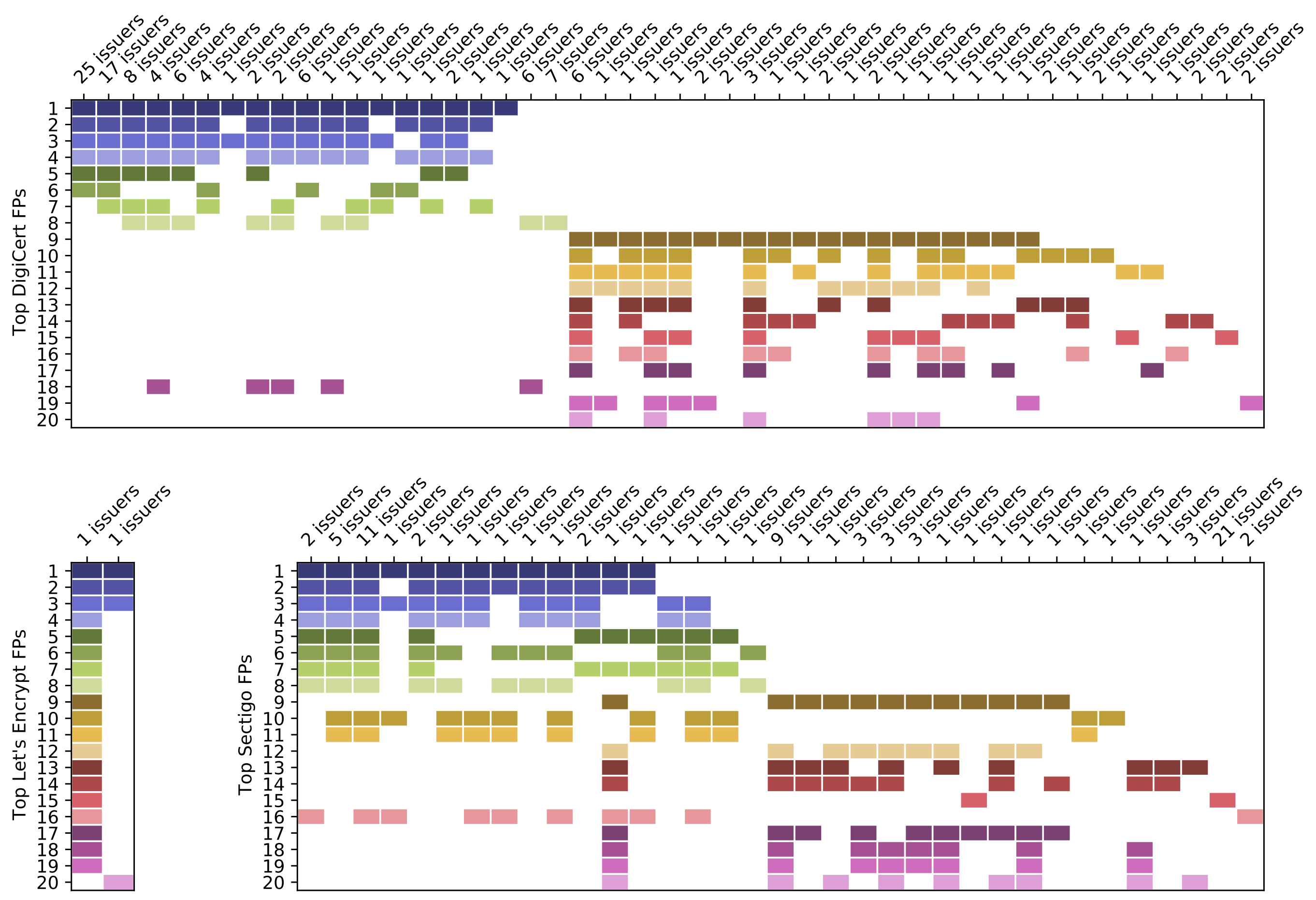


Figure 2.6: **Top Fingerprint Issuance**—The top twenty issued fingerprints for each of the top three CAs by volume reflect differences in certificate issuance. Let's Encrypt employs only two issuers that contain overlapping issuance profiles, while DigiCert and Sectigo display a multitude of issuers with varying overlap. DigiCert's issuers generate two disjoint sets of fingerprints that reflect independent CA operations (Certipost and DigiCert).

Case study. To better understand the utility of certificate fingerprints, we performed a case study examining the certificates issued by three of the top CAs by issuance volume: Let's Encrypt, Sectigo, and DigiCert. Using CCADB as a rough approximation of CA certificate control, we find that Let's Encrypt has issued certificates with 66 distinct fingerprints while DigiCert (21,856) and Sectigo (23,576) have issued over three hundred times more fingerprints. This reflects the narrowness of Let's Encrypt's automated CA operations, which only issue domain-validated certificates from a single CA software, Boulder [92]. Sectigo, DigiCert, and Let's Encrypt fingerprints are disjoint, with the exception of 11 fingerprints that are shared between one Sectigo and two DigiCert CA issuers. These overlapping fingerprints occur in CA certificates labeled "TAIWAN-CA INC.", which suggests either the same issuance software/configuration between Sectigo and DigiCert that does not appear in any other DigiCert or Sectigo certificates, or the presence of an undisclosed sub-CA under the control of Taiwan CA. Fortunately, the DigiCert certificates expired in September 2016 and Sectigo certificate in May 2020, reducing the potential danger of improper disclosure in this particular case.

Looking at the top twenty most common fingerprints and their issuers amongst these three CAs (Figure 2.6), we observe that CAs are often responsible for more than one fingerprint. To best capture the operational nature of each issuer, we compare each issuer's *issuance profile*, which is the full set of fingerprints issued by an issuer. We use the following modified Jaccard similarity metric with a heuristic threshold of 0.5 to account for issuance profiles of different sizes:

$$J_{mod}(A,B) = \frac{|A \cap B|}{min(|A|,|B|)} \tag{2.1}$$

Figure 2.6 highlights three different operational snapshots. The DigiCert issuance profiles form two disjoint clusters: issuers with fingerprints 1–8 and 18, and issuers with fingerprints 9–17 and 19–20. Manual inspection of the certificates and audits in these two clusters reveal that the first cluster belongs to the "Citizen CA," which is the PKI used for Belgium's electronic identity system. According to CA documents [93], Citizen CA is operated by Certipost, although a majority of intermediates are disclosed under DigiCert. The second DigiCert cluster contains an assortment of CA certificates operated by DigiCert itself. Sectigo's issuers display a patchwork of fingerprints with no clear clusters, which likely indicates diverse but shared issuance operations. Finally, Let's Encrypt demonstrates relatively restricted issuance across two issuers. The first issuer represents Let's Encrypt's X3 intermediate which was in operation during its introduction of support for elliptic curve public keys, pre-certificates, and OCSP Must-Staple extension. The second Let's Encrypt issuer is the retired X1 intermediate that issued a less diverse set of early certificates.

2.2.3 CA Network Infrastructure

CA network infrastructure (e.g., OCSP servers) can also hint at shared operation. We investigate the operational infrastructure used for online chain building (Authority Information Access (AIA) CA Issuer) and certificate revocation (CRL and OCSP). This infrastructure can be closely tied to the issuing CA certificate since child revocations are often signed by the issuing certificate, and AIA CA Issuer URLs provide copies of the issuing certificate. Other certificate fields that contain URLs, such as the Certificate Policies extension, do not relate directly to operational functions. In total, we extracted 2,334 FQDNs embedded within child certificates, which were composed of 991 OCSP names, 938 CRL names, and 800 AIA Issuers. We performed A-record DNS lookups for each FQDN, resulting in a total of 835 IPv4 addresses in 309 Autonomous Systems (ASes). Figure 2.7 presents the distribution of different network names and addresses across CA certificates. Approximately half of all FQDNs were only associated with a single CA certificate, which might suggest relatively isolated operations. However, the distribution of IPs have a much shorter tail, which indicates that many FQDNs share the same underlying IP addresses. To identify the shared IP addresses that are indicative of shared CA operation, we filtered out all ASes belonging to CDN networks, which can co-locate unrelated network services. After this filtering (11% IPs removed), we linked CA certificates with IPs within the same /24 subnet or with exact-match OCSP/CRL/AIA hostnames.

2.2.4 CA Audits

While certificate fingerprints and network infrastructure directly measure operational features to link CA certificates under shared control, CA audits provide a complementary data perspective. CA audits report on CA operations as examined by a third-party, professionally qualified auditor. In addition to disclosing a CA's conformance or deviation from its CA policies, CA audits often include a listing of certificates within the scope of the audit. CCADB retains a collection of all CA audits, collected from public data sources, which we downloaded, resulting in 1,266 PDF files. To convert audit PDF documents to text, we preprocessed all PDFs with Adobe Acrobat's OCR tool, since many documents contained full-page images, rather than actual text. Second, to extract text from the PDFs, we opened each file in Adobe Acrobat, selected all text, and copied it into a text file. This method was chosen after testing multiple PDF-to-text solutions (including `pdftotext`), which each had difficulty preserving the spatial relationships between text elements[2]. For the subset of PDFs that did not allow text extraction, we utilized Google Document's PDF to

[2]PDFs prioritize universally consistent rendering and specify the location of text elements, rather than their logical grouping, and many solutions extract text from left-to-right, top-to-bottom. This caused issues for tables with wrapped text columns, as an example.

	Cert FPs	**Network**	**Audit**
Cert FPs	–	28.8k (15%)	4.4k (1%)
Network	28.8k (98%)	–	47.9k (12%)
Audit	4.4k (15%)	47.9.6k (26%)	–
Shared	29.1k (99%)	72.6k (39%)	48.2k (12%)
Total	29.6k (100%)	188.2k (100%)	387.9k (100%)

Table 2.3: **Edge Co-occurrence by Perspective**—The co-occurrence rates of each technique highlight their individual precision, led by certificate fingerprints. Network infrastructure and audits account for the most CA associations, as indicated by shared edge counts.

text conversion feature. We developed a set of simple regular expressions to extract the certificate SHA-256 fingerprints that were within scope of each audit document. As part of the extraction process we observed that some CA audits only contained alternate hashes, such as SHA1, which we developed regular expressions for as well.

2.2.5 Combining Perspectives

We took a conservative approach to combining the audit, network, and certificate-based techniques discussed previously. Only certificates that were associated through at least two perspectives are considered to have common CA operation. In general, we expect well connected CA certificates—those that issue similarly fingerprinted child certificates, fall within scope of the same audits, and/or utilize the same network infrastructure—to belong to the same operational control. After applying this criteria, `Fides` generated 320 clusters containing 2,599 CA issuers, with clusters ranging in size from 2–696 CA certificates. We subsequently utilized these clusters to identify discrepancies between CCADB owner labels and `Fides`'s heuristic clusters of shared operational control.

To better understand the contribution of each perspective, we measured the co-occurrence of the three techniques (Table 2.3). Audit disclosure is the largest source of linkages between CA certificates (388k), followed by overlapping network infrastructure (188k); however, these represent noisier data that did not match other sources. The differing rates of co-occurrence for each perspective indicates their precision. Certificate fingerprints (99% overlap) have relatively high precision, whereas audit and network features are less precise ($\leq$ 39% overlap). The combination of these diverse perspectives provides a more complete picture of CA operations and can guide manual investigation of CA certificate control.

2.2.6 Limitations

The novelty of this work yields its primary limitation: little ground truth data exists to evaluate the accuracy of `Fides`. We recognize this limitation and work to reduce its impact. `Fides`'s results do not constitute ground truth data; instead we use its multi-layered aggregation of perspectives to identify higher-level certificate control inconsistencies within CCADB and point develop a new dataset that better aligns with CA operator transparency.

The certificate- and document-based properties that we observe are not guaranteed indicators of CA certificate association. It is possible, for example, that two independent CAs coincidentally issue certificates with the same ASN.1 fingerprints and share AIA Issuer/OCSP/CRL infrastructure. To help mitigate these false positive associations—independent CA operations that are wrongly associated—we take a conservative approach for each individual perspective. For example, we designed certificate fingerprints to err on the side of high precision (87,009 unique fingerprint-profiles across 4,376 issuer Subject+SPKI), and our work combines unique perspectives to strengthen confidence in common CA control.

On the flip side, there are likely false negative associations as well—common CA control that is not identified by our methods. Such instances can arise as a result of complex CA operations, CAs with a single CA certificate, omissions in audit documentation, undisclosed certificates, etc. Discovery of missing associations proves to be a challenge, but will improve as the CA community moves towards increased public disclosure and documentation.

2.2.7 Evaluation

Without ground truth data on who controls CA certificates, it is difficult to directly evaluate whether `Fides` correctly identifies their owners. Instead, we evaluate whether `Fides` is able to correctly detect ownership in the cases where there is a Mozilla Bugzilla ticket that establishes clear ownership details. Mozilla's root store policy requires timely CCADB disclosure of all technically capable CA certificates. When community members notice the lack of proper disclosure, issues are created to investigate and disclose ownership in CCADB. We identified 28 instances between May 2014 to July 2019 of delayed or invalid disclosure bugs (Appendix 5.2.2). These bug reports contain certificates that were manually examined by root store maintainers prior to CCADB disclosure and we use them as approximate ground truth data to evaluate `Fides`.

We extracted the CA certificates and issuers (Subject+SPKI) associated with each bug report and manually determined CA ownership based on bug details. We then mapped each CA certificate to its corresponding `Fides` cluster. Since `Fides` clusters are initially unlabeled—they group operationally similar certificates without identifying an CA operator—we manually labeled the clusters

	Bug Reports	**Correct**	**Issuers**	**Correct**
All Issuers	28	3 (10.7%)	150	48 (32.0%)
Active Issuers	22	7 (31.8%)	103	48 (46.6%)

Table 2.4: Fides **evaluation**—Fides's ability to identify previously undisclosed CA certificates ranges from 32% of all issuers, to 47% when excluding inactive issuers that do not issue certificates published in CT.

that contained CA certificates found in bug reports. This manual identification entails looking at the individual components of Fides: certificate fingerprints, network infrastructure, and audit statements to assign a likely CA operator for each target. We then compared the manually identified CA operator(s) with the CA disclosed in each bug report. For instance, bug report #1503638 contains a CA certificate found in a cluster of 11 certificates that are covered by WISEKey audits and utilize shared network infrastructure (e.g., `ocsp.wisekey.com`).

In total, we examined 28 bug reports that spanned 150 issuers. Fides correctly identified the operators of 32% of 150 issuers and was able to correctly label all issuers in only 3 of the 28 bug reports. This is in part because 47 unidentified issuers were *not operational* and had not issued any certificates, which prevented us from fingerprinting them. Excluding these, Fides was able to correctly identify 47% of operational CA issuers, and correctly label all issuers in 7 of 22 bug reports. For example, for #1499585, Fides is able to provide more detailed information than the bug report itself. DigiCert disclosed 14 issuers, most of which are identified by Fides as DigiCert, but one of which is correctly identified by Fides as Cybertrust Japan, an independent sub-CA. As a whole, Fides has relatively high precision, since all 48 issuers within Fides appeared in the correct cluster (i.e., no certificates for CA A occurred in a cluster dominated by CA B), but low recall, only accounting for about half of unlabeled issuers.

2.3 TOWARDS CA TRANSPARENCY

In this section, we utilize Fides to uncover CA certificates that have incorrect or misleading ownership data in CCADB. We first label CA certificates with their CCADB owner, cluster CA certificates using Fides, and detect when clusters have conflicting CCADB owners and when unlabeled CA certificates belong to a labeled cluster. We manually investigate these incongruities by examining audit documentation and operational features, and construct a new dataset that aligns more closely with CA certificate operational control.

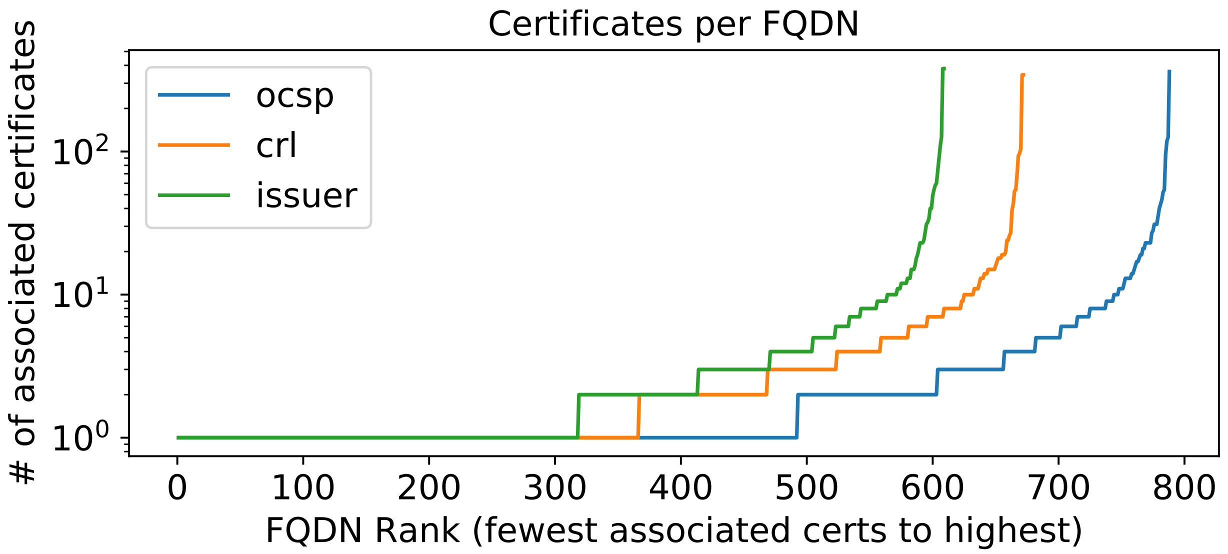
2.3.1 Labeling `Fides` Nodes

CCADB tracks individual CA certificates (i.e,. by SHA-256 fingerprint) whereas `Fides` tracks issuers by SSPKI. When we group CCADB certificates by SSPKI (Appendix B.1), we find 39 issuers (110 certificates) that map to more than one CCADB owner. 31 SSPKIs contain certificates that are revoked, expired, or properly disclosed as sub-CAs (i.e., different controlling owner). For the remaining 8 SSPKIs (20 certificates), CCADB presents control ambiguity with a single key appearing to have multiple CCADB owners. This includes 4 Let's Encrypt intermediates that are cross-signed by IdenTrust and disclosed under the IdenTrust CCADB owner, rather than the Internet Security Research Group (ISRG). The cross-signed certificates are also disclosed in the IdenTrust audit, which explicitly declares "the cross-signed certificates are not controlled by IdenTrust" [94]. This misrepresentation of CA certificate control is not a violation of disclosure policies—it merely highlights limitations of CCADB's record model. We manually identify a single CCADB owner for each of the ambiguous issuers by examining audits and certificate subjects. As part of this process, we discovered an improperly disclosed Subordinate CA by Camerfirma [95], which added to the growing list of compliance issues that recently prompted discussion about Camerfirma's fitness for Mozilla's root store [96].

After resolving the 39 SSPKI ownership conflicts (correcting 64 certificates), we find 557 CA certificates not present in CCADB that share an SSPKI with a CCADB-labeled certificate. Because a shared SSPKI represents the same cryptographic keying material, we expand CCADB labels to these CA certificates. We characterize these newly labeled certificates in Section 2.3.4. Having resolved SSPKI ownership inconsistencies, we next identify more subtle discrepancies and absences in two ways.

2.3.2 Multi-operator Clusters

The presence of multiple CCADB owners within a single `Fides` cluster points to likely misalignment between CCADB labels (including sub-CA reporting) and CA certificate control as automatically inferred by `Fides`. We identified eleven such instances (Figure 2.5) and then manually inspected each cluster to determine the root causes of mismatch, described below. Two are false positives, and the remaining nine clusters—comprised of 728 issuers across 581 certificates—point to a range of discrepancies between CCADB labels and CA control. By resolving the issues described below, we correct the labels for 125 issuers and 136 CA certificates.



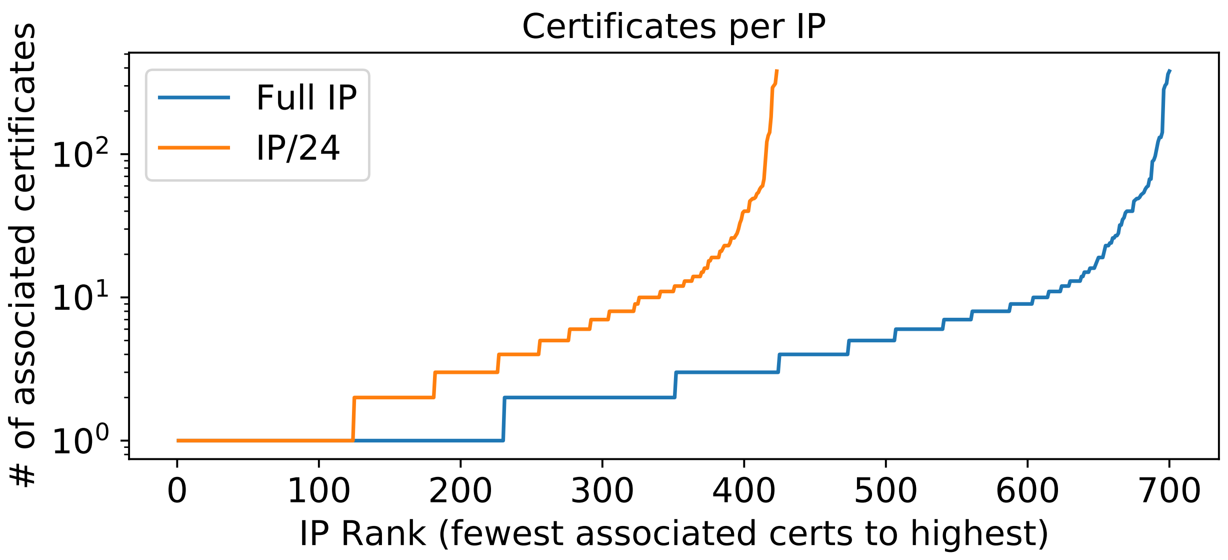


Figure 2.7: **Certificate URL and IPs**—Distribution of network infrastructure used for OCSP, CRL, and AIA Issuer URLs in CA certificates.

Cluster	CA1: # issuers (certs)	CA2: # issuers (certs)	Shared Features					Outcome
			CRL	OCSP	AIA	Cert FP	Audit	
2	🟩 Sectigo: 313 (382)	🟥 Web.com: 6 (14)	✓	✓	✓	✓	✓	White-label sub-CA.
4	🟥 DigiCert: 109 (110)	🟩 Certipost: 19 (21)	✓	✓	✓	✓	✓	Undisclosed control.
6	🟩 GlobalSign: 75 (118)	🟩 Google: 23 (33)	✓	✓	✓	✓	✓	False positive.
21	🟩 GoDaddy: 9 (19)	🟩 Amazon: 2 (7)	✓	✓	✓	-	✓	False positive.
60	🟩 Digidentity B.V.: 3 (4)	🟥 PKIoverheid: 2 (2)	-	✓	-	-	✓	Undisclosed control.
64	🟥 DigiCert: 2 (4)	🟥 Sectigo: 1 (1)	✓	-	-	✓	-	Undisclosed third-party.
67	🟩 TC TrustCenter: 2 (3)	🟥 DSV GmbH: 1 (1)	-	-	✓	✓	-	Undisclosed control.
94	🟩 Deutsche Telekom: 2 (2)	🟥 DigiCert: 1 (1)	-	✓	-	✓	-	Undisclosed control.
183	🟩 StartCom: 1 (1)	🟥 Certinomis: 1 (1)	-	✓	-	✓	-	Undisclosed control.
212	🟩 E-Tugra: 1 (1)	🟥 e-tugra: 1 (1)	-	✓	-	✓	-	Clerical error.
252	🟩 E-Tugra: 1 (1)	🟥 e-tugra: 1 (1)	-	✓	-	✓	-	Clerical error.

Table 2.5: **Multi-operator clusters**—11 clusters have clashing CCADB labels. Green/red cells represent correct/incorrect match between CCADB owner labels and CA operational control. CCADB labels misrepresent the control of 125 issuers across 136 certificates.

White-label sub-CA. Cluster 2, the second largest cluster, contains CA certificates belonging to both Sectigo and Web.com. Web.com is properly disclosed as a sub-CA of Sectigo, but unlike other disclosed sub-CAs (e.g., Apple, which is also a sub-CA of Sectigo), Web.com exhibits the same operational features as its parent CA, Sectigo. Several Web.com and Sectigo issuers share the same AIA infrastructure (`crt.usertrust.com`) and their OCSP/CRL infrastructure utilize identical IP addresses. Furthermore, 391 out of 540 Web.com issuance fingerprints overlap with Sectigo generated fingerprints, suggesting a shared certificate issuance pipeline. Audit reports corroborate these findings, indicating that Sectigo controls cross-signed certificates for Web.com and that both CAs operate in the same locations globally [97, 98]. Most sub-CAs operate independent of their parent CA, but Web.com appears to utilize a white-label CA service provided by Sectigo [71]. `Fides` automatically spotlights the shared operations of Sectigo and Web.com, which should be treated as closely intertwined participants in the CA ecosystem, despite their differentiation in CCADB. We further update the `Fides` dataset to indicate that all Web.com CA certificates are effectively operated by Sectigo.

Undisclosed control. Six clusters with multiple CCADB labels constitute undisclosed CA certificate control. Cluster 4, which contains CA certificates used for Belgium's electronic ID cards, contains the largest number of misrepresented CA certificates. Although three root certificates are disclosed as a sub-CA of DigiCert called Certipost NV/SA (which runs the Belgian Citizen CA), all of the intermediates under those roots contain only a DigiCert CCADB label. All operational features, including third-party audits [99], point to Certipost control of these CA certificates. Cluster 60 also displays incomplete sub-CA disclosure: 2 PKIoverheid intermediates are disclosed as a Digidentity sub-CA, but their child intermediates are labeled as PKIoverheid, contradicting audit records [100].

Cluster 67 contains two root certificates that CCADB labels as TC TrustCenter, and one root certificate that CCADB labels as DSV GmbH. All three utilize the same `www.trustcenter.de` AIA issuer, contain the name "TrustCenter", and generate a shared set of globally-unique issuance fingerprints. The evidence suggests that TC TrustCenter controls all three roots. Cluster 94 and 183 represent similar cases between Deutsche Telekom / DigiCert and StartCom / Certinomis. As previously discussed in Section 2.2.2, Taiwan CA (TWCA) appears to operate cluster 64 based on certificate fingerprinting. The issuers in cluster 64 also share CRL infrastructure (`sslserver.twca.com.tw`), further suggesting TWCA operated as an undisclosed sub-CA of both Sectigo and DigiCert. In this instance, a third-party not indicated by CCADB labels actually operated the CA certificates within the cluster.

Clerical error. Clusters 212 and 252 provide an example of CCADB clerical error. CCADB contains two distinct variations of the Turkish SSL provider, E-Tugra (10 CA certs) and e-tugra (6 CA certs), which suggests two distinct CCADB administrator accounts for a single CA. The explanation for this behavior is unknown. While this is the only administrative quirk that emerges from `Fides` cluster analysis, a manual investigation of CCADB's Subordinate CA owners reveals further clerical quirks. 7 sub-CAs contain inconsistent naming such as alternate spellings (e.g., "Quo Vadis" versus "QuoVadis") or syntactic differences (e.g., "DigitalSign – Certificadora Digital, SA" versus "DigitalSign –Certificadora Digital, S.A."). These may seem like minor details that manual inspection can clarify, but we note that CAs may have very similar names, as is the case with SSLCOM and SSL.com, and sloppiness can lead to misidentification.

False positives. `Fides` falsely grouped two clusters of CA certificates. The first, cluster 6, contained CA issuers labeled by CCADB as GlobalSign (76 issuers) and Google (23 issuers). `Fides` detected shared OCSP infrastructure, audits, and certificate fingerprints between two GlobalSign issuers[3] and two issuers[4] labeled as Google Trust Services (GTS) by CCADB. In actuality, these CAs are currently operated independently, but `Fides` mistakenly clusters GTS and GlobalSign issuers because they were historically operated by GlobalSign. GTS acquired two GlobalSign roots in 2016 [101], but `Fides`'s chronology unawareness leads to the false positive grouping of CA operation. A very similar root acquisition occurred between Amazon Trust Services (ATS) and GoDaddy [102], leading to the second false positive clustering. To better address these scenarios, future work can incorporate chronologically differentiated operational profiles to detect transitions in certificate control.

2.3.3 Minority unlabeled clusters

We identified 17 `Fides` clusters (Table 2.6) where a minority of nodes are unlabeled, and a supermajority (more than 70%) of nodes share the same CCADB owner label. In total, `Fides` labeled 94 certificates spanning 84 issuers. Due to insufficient audit data and CCADB metadata for these newly-labeled CA certificates, we cannot properly assess the accuracy of these new labels, and false positives such as those identified in Section 2.3.2 could exist. To reduce these possibilities, we chose a conservative 30% threshold of unlabeled nodes. `Fides`'s CA operator labels represent a best-effort guess for CA certificates that would otherwise have no CA control information available. We further examine these previously unlabeled certificates in Section 2.3.4, alongside the SSPKI expanded labels from Section 2.3.1.

[3]*GlobalSign PersonalSign 2 CA - SHA256 - G3* and *GlobalSign EC Administration CA2*
[4]*GlobalSign ECC Root CA - R4* and *GlobalSign EC Administration CA1*

Cluster	**Primary Operator**	**Unlabeled Iss. (Certs)**	**Unlabeled %**
2	Sectigo	7 (8)	2.1%
3	DigiCert	7 (8)	3.8%
4	Certipost s.a./n.v.	41 (41)	24.3%
5	DigiCert	7 (10)	6.2%
7	Asseco	3 (4)	4.5%
8	HARICA	2 (2)	3.6%
13	Entrust	2 (2)	8.3%
15	SwissSign AG	2 (2)	10.5%
16	SecureTrust	1 (2)	5.6%
28	Gov. of Hong Kong	1 (1)	11.1%
36	DigiCert	2 (2)	28.6%
38	DigiCert	2 (2)	28.6%
41	IdenTrust	1 (1)	14.3%
42	Cybertrust Japan	2 (3)	28.6%
57	GlobalSign	1 (4)	20.0%
67	TC TrustCenter	1 (1)	25.0%
69	KIR S.A.	1 (1)	25.0%
17 clusters	*14 operators*	*83 (94)*	–

Table 2.6: **Minority unlabeled clusters**—94 CCADB-undisclosed certificates appear in 17 clusters with a super-majority (¿70%) of known issuers. Undisclosed, `Fides`-clustered CA certificates occur across a range of CA operators.

2.3.4 CA operator dataset

Building off of CCADB-labeled clusters, and merging in our investigation of owner ambiguities (Section 2.3.1) and discrepancies between CCADB labels and `Fides`'s operational clusters, we develop a new dataset that more accurately describes the organizations that control each CA certificate. The dataset corrects the administrative CCADB labels of 241 CA certificates by resolving multiple CCADB owner conflicts within a single SSPKI or `Fides` cluster. Through SSPKI and cluster expansion, the dataset also extends coverage to 651 CA certificates beyond CCADB disclosure, which is limited by CA self-reporting and the fact that not all root stores require CA certificate disclosure. In total, `Fides` improves or extends coverage for 208 trusted CA certificates, or 6.2% of all 3,338 CA certificates trusted by Microsoft, Apple, or NSS (Table 2.7). We hope that this dataset, which we provide open-source [64], enables improved CA research and CA trust decision making. Below, we investigate the potential explanations for `Fides`'s findings and CCADB's shortcomings.

`Fides` **Relabeled** Guided by manual analysis, `Fides` identifies 241 CA certificates where CCADB labels disagree with operational features. The conflicting DigiCert/Certipost cluster accounts for nearly half (114) of these instances. Excluding these certificates, we find twenty CAs that act as the CCADB administrator for a CA certificate they do not operate, indicating that for many CAs, CCADB owner labels signal administrative responsibility rather than operational control. In many cases, these CA certificates are disclosed as sub-CAs, but disclosure is often incomplete, as detailed in Section 2.5. About a quarter (64 out of 241) of `Fides` relabeled CA certificates resulted from conflicting CCADB owners for a shared SSPKI. Although conflict resolution requires manual investigation, CCADB could add an automated notification or require a sub-CA label when a single SSPKI maps to certificates with multiple CCADB owners.

`Fides` **Newly Labeled** `Fides` automatically assigned labels to 651 CA certificates not present in CCADB. In the absence of ground truth data for newly labeled certificates, we tracked ten CA certificates that were added to CCADB between July 2020 and February 2021. All ten CA certificate had CCADB labels that matched the independently generated `Fides` labels (including four Web.com /Sectigo certificates). As an additional confirmation of these new `Fides` labels, we examined the 62 CA certificates that appeared in audits. We manually identified the CA operator in each audit and found that 60 out of 62 (96.7%) `Fides`-labeled operators match audit records. The two certificates with erroneous labels occur because two DigiCert cross-signs of MULTICERT certificates contain a DigiCert CCADB label. In this instance, `Fides` propagates a CCADB label that does not match CA certificate control. Future work classifying the CAs described in CA audits could provide additional consistency checks to further improve `Fides`'s accuracy.

Why were these newly labeled certificates not included in CCADB? Only 75 certificates are trusted by NSS, and only 20 had not expired before February 2017 when NSS mandated CCADB disclosure [103]. NSS does not require disclosure of technically constrained CA certificates (6) or those without TLS server authentication capabilities, which applies to the remaining 13 certificates. `Fides` does not discover improperly undisclosed NSS-trusted CA certificates, suggesting general compliance with the Mozilla Root Store disclosure policies. However, as described in Section 2.3.1 we do discover improperly disclosed CA certificates.

For CA certificates trusted by Apple or Microsoft, `Fides` expands coverage of CA operators by 209 certificates. 141 of these certificates are expired, limiting their utility, but can provide data for historical CA behavior studies. The 68 remaining certificates improve public understanding of active CA operation, especially for the six CA certificates (4 Sectigo, 1 DigiCert, 1 TrustFactory) with unconstrained server authentication capabilities. Because each unconstrained CA certificate is a single point of widespread failure (i.e., a compromised CA certificate can impersonate most

	Total Iss. (Certs)	**Trusted Iss. (Certs)**	**Valid Iss. (Certs)**
CCADB	4,845 (6,195)	2,673 (2,961)	3,457 (4,077)
Relabeled	189 (241)	85 (90)	103 (121)
New label	404 (651)	90 (115)	130 (164)
`Fides`	4,928 (6,846)	2,707 (3,076)	3,490 (4,241)

Table 2.7: **CCADB/`Fides` Comparison**—`Fides` yields a CA operator dataset that corrects CA operator labels for 90 trusted CA certificates from 85 issuers, and extends coverage by 115 trusted CA certificates. `Fides` improves/increases coverage for 6.1% of all 3,338 trusted CA certificates.

domains[5]), comprehensive transparency of the CA certificates wielded by each CA can help attribute suspicious behavior or mitigate more serious issues when they occur.

2.4 DISCUSSION

While `Fides` can detect inconsistencies between CCADB ownership labels and operational practices, it is not a long-term solution. Its heuristics are not perfect, and while its precision is high, its recall is low. We hope that `Fides` sheds light on the poor state of affairs, quantifying how certificate subjects poorly reflect CA ownership and showing how CCADB does not currently address controlling ownership. True transparency requires changes to existing CA procedures and root store requirements. Below, we explore potential solutions:

CCADB Structured Data. At the moment, CCADB plays a critical role in the PKI ecosystem: it provides a mechanism for CA certificate data to update independent of the actual certificate itself. CCADB provides mutability to CA certificates. Because the frequency of CA certificate control changes outpaces the frequency of CA certificate replacement, current CA certificates must divorce their names (stored in the certificate) from their identity (stored outside of the certificate). CCADB is a natural location to track who controls each CA root and intermediate certificate. While in some cases we can infer certificate control from CCADB record owners and uploaded audits, the data is not easily accessible. Adding explicit fields for ownership details would allow both root store operators and researchers to better track CA behavior, and would additionally provide data compare against regular `Fides` runs. This proposal is the simplest to implement, but would require careful auditing and consistent monitoring to protect against error-prone or even nefarious self-reporting. User agents can also enforce more stringent CCADB inclusion policies to help remove trust dependencies on CAs that have refused to submit details to CCADB.

[5]Exceptions for HSTS, preload, and CAA.

Increased Intermediate Restrictions. While trust anchors are long-lived and shipped with user agents, intermediate CA certificates do not need to be. User agents can require that intermediate CA certificates contain up-to-date ownership details, similar to the requirements for Extended Validation (EV) certificates, and to restrict their change of ownership. Because leaf certificates are signed by intermediates rather than a trust anchor, this would allow users to always identify the entity that signed the certificate used when accessing a website. User agents could further limit the validity period of intermediate certificates to disincentivize transfer of ownership and reduce the impact of changes in certificate control.

Reconsider Root CA Labels. Today, user agents already ignore some details about trust anchors, including their validation periods. We should consider whether we should also ignore included trust anchor subject names and to instead ship these details with the root store. As it stands, labels on roots are misleading for a significant fraction of CAs, and browser-supplied labels could provide more up-to-date ownership details (e.g. as extracted from CCADB).

These proposals are orthogonal to the development and deployment of `Fides`, which can help identify user errors and suspicious CA practices. Future development of `Fides` can verify the consistency between CA documentation/audits claims and externally measurable behavior. For example, by extending `Fides` to include the IP addresses from which CAs deliver certificates to Subscribers, we could automatically check the accuracy of the operational locations disclosed in CA documentation/audits. Further development of `Fides`'s certificate fingerprinting techniques can also identify the CA software that different CAs use, leading to better discovery and remediation of certificate issuance problems, such as the widespread usage of 63-bit serial numbers due to an EJBCA bug [104].

2.5 SUMMARY

In this chapter, we utilized a combination of CA operational perspectives (i.e., network infrastructure, issuance infrastructure, audits) to uncover the CAs that actually operate CA certificates. Our transparency efforts yield an open-source set of 6,846 CA certificates labeled with their CA operator. Initial application of this dataset identified a mis-attributed CA certificate that contributed to the removal of the offending CA (Camerfirma) from Mozilla's root store. We believe that future applications and research will improve the overall security of root store trust.

CHAPTER 3: TLS TRUST ANCHOR ECOSYSTEM

Having identified the entities who operate CA certificates, we now examine decision making in trust management. Trust decisions occur in two different situations. First, root program operators must choose which CAs to include in their root store. Second, TLS software developers must decide whether to implement their own root store or choose an existing root store to trust. We begin by enumerating the most popular TLS user agents and tracking the root stores they depend on. We then expand this baseline with other popular TLS libraries and operating systems to capture a broader view of root store trust. By clustering root stores over time, we converge on the three independent root programs that underlie the majority of TLS authentication trust. Finally, we compare these independent foundations as well as the derivative root stores to understand their different flavors of trust and implications for security.

3.1 BACKGROUND

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 for *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 and 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 [5] for a broader overview of TLS.

All user agents that utilize the PKI rely on a trusted, or a set of trusted, "root" CA(s) that comprise its trust anchor / 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 in the Baseline Requirements (BRs) [39], which is maintained by the CA/Browser Forum, a consortium of CAs and parties that rely on TLS PKI. Each root store operator may also have custom requirements for including or removing a CA from its root store. For instance, Mozilla runs a relatively rigorous policy that requires roots to publicly disclose unconstrained intermediates in Common CA Database (CCADB) [74], 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 identity. 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. For root certificates, the BRs require KU to specify certificate signing and certificate revocation list (CRL) signing, but EKU must be blank. EKU purposes are only specified on intermediate CA certificates. CAs dictate the constraints found within a certificate, but end entities that utilize certificates 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.

As suggested by the presence of EKU, 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 as well as secure email transport (STARTTLS).

3.2 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. Table 3.1 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 13% of user agents that we have unknown or no coverage for 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.

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

Root store	From	To	# SS	# Uniq	Data source	Details
Alpine	2019-03	2021-04	42	7	docker [105]	`/etc/ssl/cert.pem` or `/etc/ssl/ca-certificates.crt`
AmazonLinux	2016-10	2021-03	43	15	docker [106]	`ca-trust/extracted/pem/tls-ca-bundle.pem` aggregate file of root certs
Android	2016-08	2020-12	14	7	source [107]	List of root certificate files.
Apple	2002-08	2021-02	109	43	source [108]	Both macOS and iOS. `certificates/roots` directory of files
Debian	2005-05	2021-01	39	29	source [109]	`/etc/ssl/certs` and `/usr/share/ca-certificates`, directory of cert files
Java	2018-03	2021-02	7	7	source [110]	`make/data/cacerts` JKS file that has migrated over time
Microsoft	2006-12	2021-03	86	70	update file [111]	`authroot.stl` updates roots, trust purpose, addl. constraints
NodeJS	2015-01	2021-04	16	11	source [112]	`src/node_root_certs.h` list of certificates
NSS	2000-10	2021-05	225	63	source [113]	`certdata.txt` stores roots, trust purpose, additional constraints
Ubuntu	2003-10	2021-01	38	29	source [114]	`/etc/ssl/certs` and `/usr/share/ca-certificates`, directory of cert files

Table 3.2: **Dataset**—Root store history of 619 total snapshots (SS) for ten root providers: seven OS, three library.

We supplement the popular user agent dataset by expanding our data collection process (Appendix 5.2.2) to consider a total of nine popular mobile and desktop operating systems, which all 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 12 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 3.2 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" [115]. 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 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 contained within `certdata.txt`, due to their complexity and technical considerations [116]. 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` [116]. As discussed in Section 3.2, 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, a well specified standard for root CAs. As part of that policy, NSS maintainers must work with the community [117, 118] 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 [80, 119, 120]. 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 [121], and uncovered WoSign's surreptitious ownership of StartCom [122]. 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 [123]) 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 specify the purposes for which a certificate is trusted or distrusted, as well as other more nuanced trust, such as those discussed in Section 3.4.3. 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 [111].

Apple Since at least 2005 [124], 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 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.

Linux distributions Most Linux distributions derive their trust anchor stores from NSS. However, rather than use NSS's `certdata.txt`, 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.

[1]Whether applications utilize these purpose-based roots is beyond the scope of this study.

Android Android maintains its own trust anchor store [107]. 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 [125] 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 developers of Java, operates a root program [126] 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 [127], 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. Modern Opera has migrated to Chromium and utilizes system root stores.

Electron Electron is an application development framework 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 [128], Electron can rely on either Node's root store, or the system root store, which Chromium defaults to.

Data Collection & Limitations

The root store providers described above manage and publicly release their trust anchors in different formats. We parse these formats and consolidate 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

[2]Apple policies prohibit custom root policies

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 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 [54] and locally installed AV / monitoring software [129]. 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[3]. 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 handle exceptional cases, rather than commonplace scenarios. When we discuss specific root store inclusions/removals in Sections 3.4 and 3.5, we make a best-effort attempt to account for any trust logic external to the root store itself.

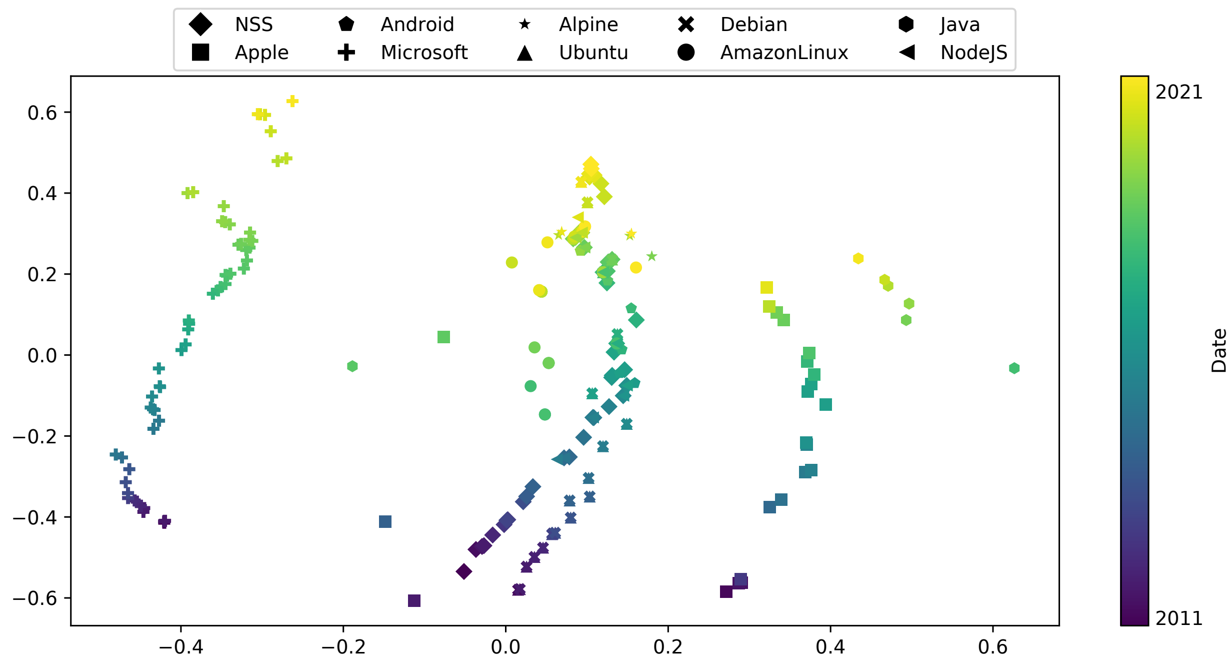


Figure 3.1: **Root Store Similarity**—Performing MDS on the Jaccard distance between root store providers from 2011–2021 illustrates four distinct clusters of roots. From left to right: Microsoft, NSS-like, Apple, Java.

3.3 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 Amazon-Linux / 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 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

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

implemented by Python's `sklearn` library [130].

From Figure 3.1, we can see four clusters emerge from the eleven root store providers in our dataset. These correspond (from top to bottom) with Java, Apple, NSS/Linux/NodeJS, and Microsoft. 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), which indicates that even though each root store family has evolved, they have not converged or diverged drastically. 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 3.5.

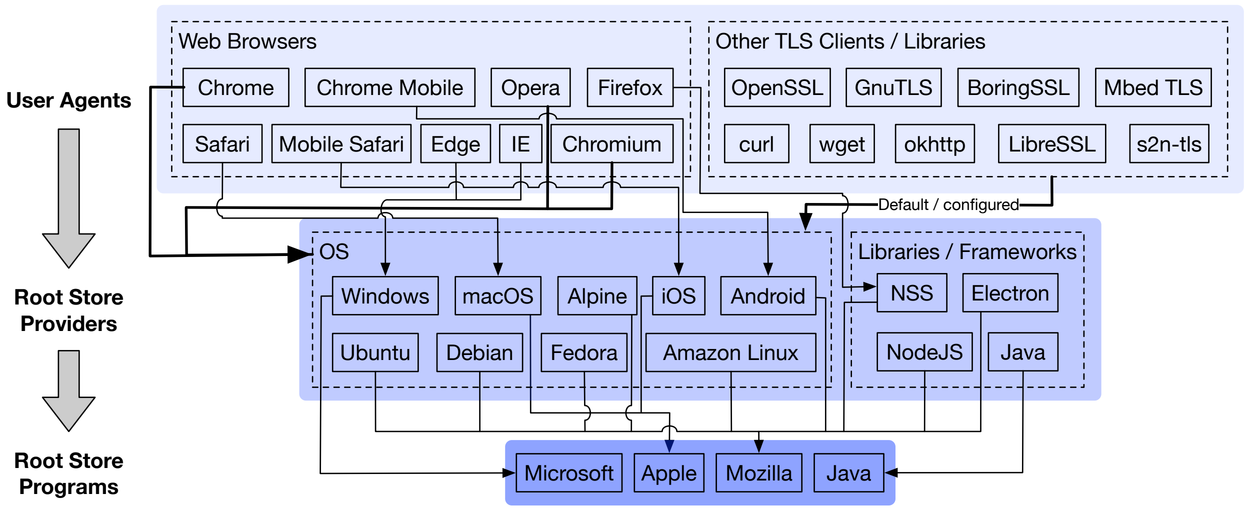


Figure 3.2: **Root Store Ecosystem**—The TLS root store ecosystem is an inverted pyramid, with a majority of clients trusting one of four root families.

From tracing the top 200 user agents to their root store family, we find that NSS (34%), Apple (23%), and Windows (20%) together account for a majority of the user agents. As shown in Figure 3.2, the root store ecosystem is an inverted pyramid, built primarily upon these root programs.

3.4 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 3.3: **Root store hygiene**—The average number of roots/expired roots in each root store snapshot and removal dates for trusted MD5/1024-bit RSA certificates.

3.4.1 Root store management

As a proxy for responsible root store management, we examine three metrics (Table 3.3): 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.

3.4.2 Exclusive Differences

To better quantify the differences between each root store family, we characterize the root CAs that are unique to each. Appendix 5.2.2 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 a 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 complements 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 lacks the 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 trust in a Government of Venezuela root is more questionable. This root was rejected from NSS due to the CA's position as a super-CA [131] 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[4]. One such subordinate CA, PROCERT, gained entry into NSS in 2010, but was subsequently removed after repeated transgressions [132]. This issuer was also trusted by Microsoft, but only for email, until it was blacklisted in 2020.

Microsoft Microsoft contains 30 exclusive root certificates. One-third are roots that attempted and failed the NSS inclusion process, either due to NSS rejection (6 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 trust 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 20 Microsoft-exclusive roots reflect a more innocuous collection of CAs with ongoing NSS inclusion evaluation (6 roots), recently accepted in NSS (3 roots), minimal Certificate Transparency presence (5 roots), and miscellaneous discrepancies (6 roots).

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

Root store	# Certs	Trusted until	Lag (days)	Root store	# Certs	Trusted until	Lag (days)
DigiNotar [133]		**2011-10-06**		**WoSign [135]**		**2017-11-14**	
Microsoft	1	2011-08-30	-37	*Debian/Ubuntu*	4	2017-07-17	-120
Apple	1	2011-10-12	6	Microsoft	4	2017-09-22	-53
Debian/Ubuntu	1	2011-10-22	16	*Android*	4	2017-12-05	21
CNNIC [134]		**2017-07-27**		*NodeJS*	4	2018-04-24	161
Apple	2	2015-06-30	-758	*AmazonLinux*	4	2019-02-18	461
Android	1	2017-12-05	131	**PSPProcert [132]**		**2017-11-14**	
Debian/Ubuntu	2	2018-04-09	256	*Debian/Ubuntu*	1	2018-04-09	146
NodeJS	2	2018-04-24	271	*NodeJS*	1	2018-04-24	161
AmazonLinux	2	2019-02-18	571	*AmazonLinux*	1	2019-02-18	461
Microsoft	2	2020-02-26	944	**Certinomis [136]**		**2019-07-05**	
StartCom [135]		**2017-11-14**		*NodeJS*	1	2019-10-22	109
Debian/Ubuntu	3	2017-07-17	-120	*Alpine*	1	2020-03-23	262
Microsoft	2	2017-09-22	-53	*Debian/Ubuntu*	1	2020-06-01	332
Android	3	2017-12-05	21	*Android*	1	2020-09-07	430
NodeJS	3	2018-04-24	161	*AmazonLinux*	1	2021-03-26	630
AmazonLinux	3	2019-02-18	461	Apple	1	2021-01-01*	577
Apple	3	2021-02-01*	1,175	Microsoft	1	2021-03-03*	607

**Still trusted*

Table 3.4: **High severity removals**—Comparison of root store responses to high severity NSS removals.

3.4.3 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 span 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 prompted 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 5.2.2. Although Mozilla provides a Removed CA Report [137], this data misses 87 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.

Table 3.4 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 [138], 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 [139], and Apple follow suit on September 9 [140]. Although the up-to-the-hour response delay in such incidents is important, our dataset lacks the resolution (see Section 3.2) 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 [141]. Apple took an alternate approach—they removed the CNNIC root in

2015, significantly before other root programs, but whitelisted 1,429 leaf certificates [142]. 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 [135]. 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 StartCom roots*, 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. StartCom delayed disclosure of these cross-signs by 111 days. Apple and Microsoft have not removed this certificate.

3.5 NSS DERIVATIVES

In NSS documentation, Mozilla explicitly 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" [115]. In this section, we track the behavior of NSS-derivative root stores to characterize their security practices surrounding NSS root store adoption. We 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.

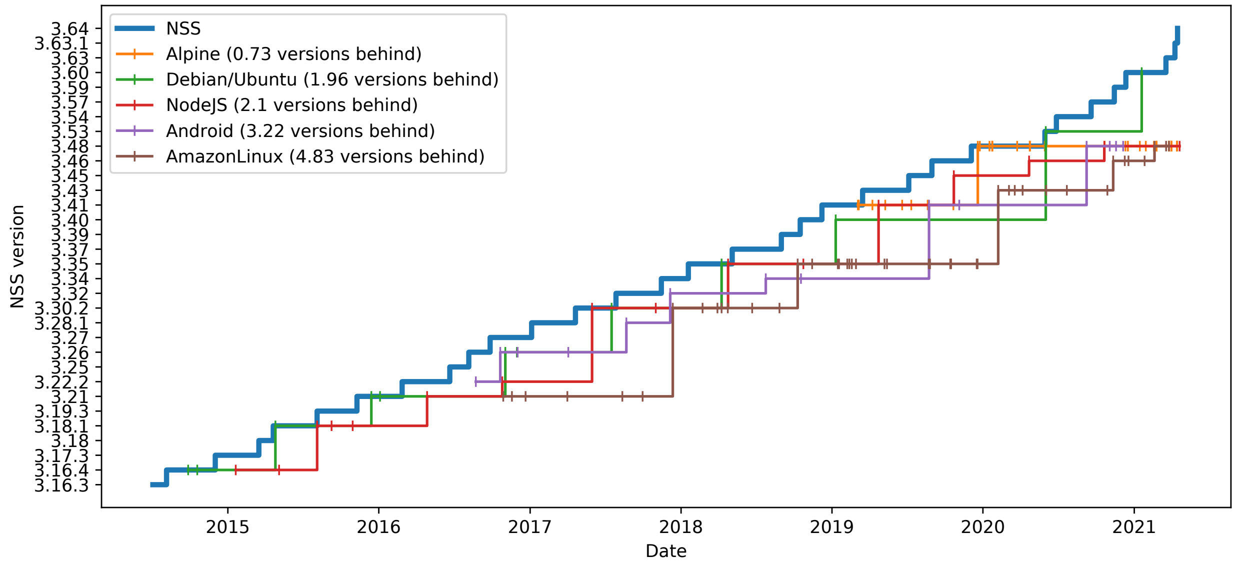


Figure 3.3: **NSS derivative staleness**—No derivative root stores match NSS's update regularity. Alpine Linux maintains closest parity to NSS, while AmazonLinux, on average, lags more than four substantial versions behind.

3.5.1 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 match. Instead, we use Jaccard set distance and find the closest NSS version match for each derivative root store snapshot. Figure 3.3 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 3.3 represents a mainline version update for each derivative, which suggests

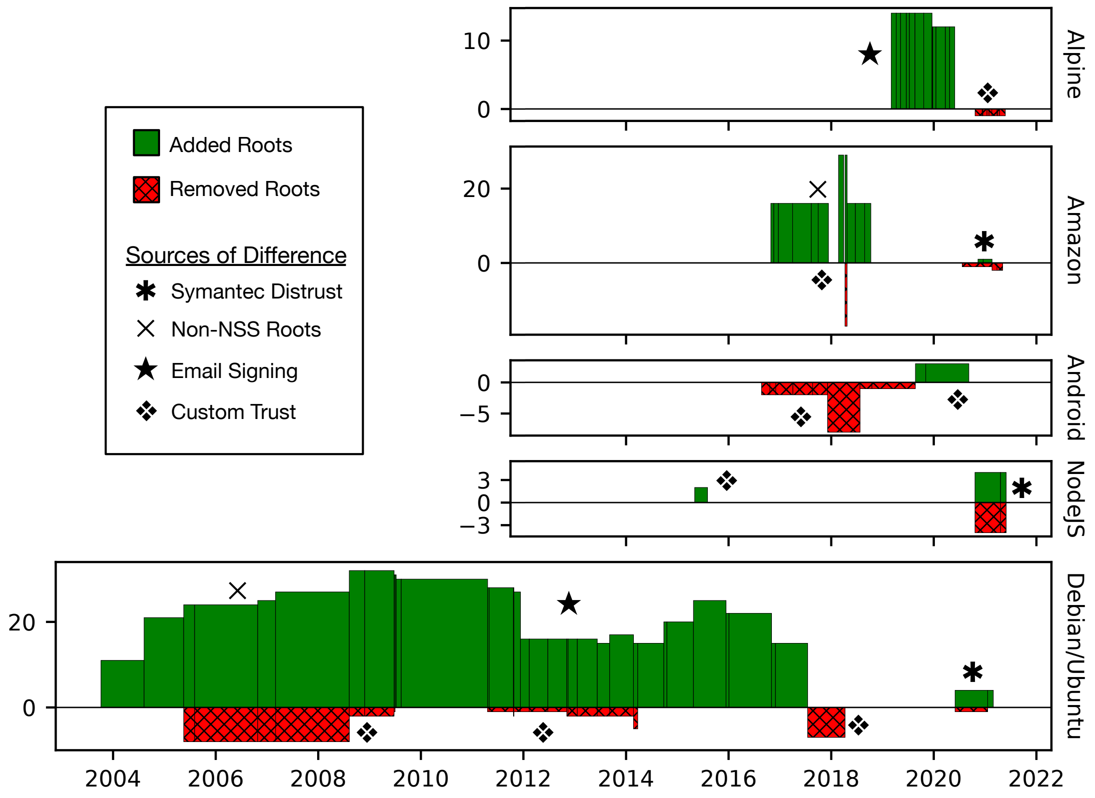


Figure 3.4: **NSS derivative diffs**—The number of added/removed root certificates for each NSS derivative indicates that all deviate from strict NSS adherence.

that some derivative version updates ignore potential NSS updates, especially for Amazon Linux and Alpine.

3.5.2 Derivative Differences

Figure 3.4 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 practices and misunderstanding of 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. This distrust event 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 [143] to distrust subscriber certificates issued after a certain date. In 2020 [144], 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: still trusted until expiration, and not trusted. However, none of the NSS derivatives had such a mechanism and were forced 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 version 53 and has continued to apply subsequent NSS updates. Unfortunately, NSS 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—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 [145]. Unfortunately, this premature full distrust led to so many user complaints [146] 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 [147]. This is 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), 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), CACert(3). Of these CAs, only DCSSI has ever had a root included in NSS. While we do not trace the inclusion reasons for all these 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 [148] and other Linux distributions [149] for lack of audits. These practices are a significant departure from NSS inclusion policies and potentially put Ubuntu/Debian users are 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 significantly. 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 when the changes are not necessitated by falling behind the most recent NSS update.

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 [150]. These additions are likely necessitated by impact to end users of the derivative root store, which differ from NSS's users and risk calculus.

3.6 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, we expect existing pain points in the TLS root store ecosystem to 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 3.5.1), 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 [108, 151] that account for more nuanced root store trust. Given that multitudes more root stores likely already rely on NSS[5], we hope that future work helps transition derivatives and new root stores to more modern formats.

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. 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-derived 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 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).

3.7 SUMMARY

In this chapter, we discovered the previously-unknown provenance of root store trust in TLS authentication. We collected the root stores for over 75% of the top HTTP user agents seen at

[5]Searching `certdata.txt` in Github yields over 203k code results.

a global CDN provider, and found only three independent root programs Microsoft, Apple, or Mozilla that underlie global TLS trust. Many root store providers (e.g., Node, Linux variants) copy their trust, and they exclusively copy from Mozilla's NSS. NSS dependence is manually and haphazardly implemented; this leads to questionable trust customizations, mistaken trust, and stale trust in a large number of systems.

CHAPTER 4: IDENTIFYING TLS INTERCEPTION

TLS root stores are mutable and can deviate from the default trust anchors installed by an operating system or other root store provider. They can be modified by system administrators, end-users, or any software with sufficient permissions. Unfortunately, no existing techniques provide a scalabe means to detect and analyze the customized trust of real-world root store deployments. To address one facet of this issue, we look at TLS interception. We develop a new method for distinguishing intercepted traffic from normal traffic and apply it to three real world datasets that consist of 7.75B TLS connections. We also evaluate 29 interception product versions to understand their impact on network security.

4.1 BACKGROUND

In this section, we provide a brief background on HTTPS interception and describe the aspects of HTTP and TLS that are relevant to our fingerprinting techniques. We refer the reader to RFC 5246 [152] for a detailed description of TLS.

4.1.1 TLS Interception

Client-side software and network middleboxes that inspect HTTPS traffic operate by acting as transparent proxies. They terminate and decrypt the client-initiated TLS session, analyze the inner HTTP plaintext, and then initiate a new TLS connection to the destination website. By design, TLS makes such interception difficult by encrypting data and defending against man-in-the-middle attacks through certificate validation, in which the client authenticates the identity of the destination server and rejects impostors. To circumvent this validation, local software injects a self-signed CA certificate into the client browser's root store at install time. For network middleboxes, administrators will similarly deploy the middlebox's CA certificate to devices within their organization. Subsequently, when the proxy intercepts a connection to a particular site, it will dynamically generate a certificate for that site's domain name signed with its CA certificate and deliver this certificate chain to the browser. Unless users manually verify the presented certificate chain, they are unlikely to notice that the connection has been intercepted and re-established.[1]

[1]Contrary to widespread belief, public key pinning [153]—an HTTPS feature that allows websites to restrict connections to a specific key—does not prevent this interception. Chrome, Firefox, and Safari only enforce pinned keys when a certificate chain terminates in an authority shipped with the browser or operating system. The extra validation is skipped when the chain terminates in a locally installed root (i.e., a CA certificate installed by an administrator) [154]. Internet Explorer and Edge do not support key pinning [155].

4.1.2 TLS Feature Negotiation

TLS clients and servers negotiate a variety of protocol parameters during a connection handshake [152]. In the first protocol message, Client Hello, the client specifies what TLS version and features it supports. It sends ordered lists of cipher suites, compression methods, and extensions—which themselves frequently contain additional parameters, such as supported elliptic curves and signature algorithms. The server then selects a mutually agreeable choice from each list of options. This extensibility facilitates the continuing evolution of features and provides adaptability in the wake of new attacks.

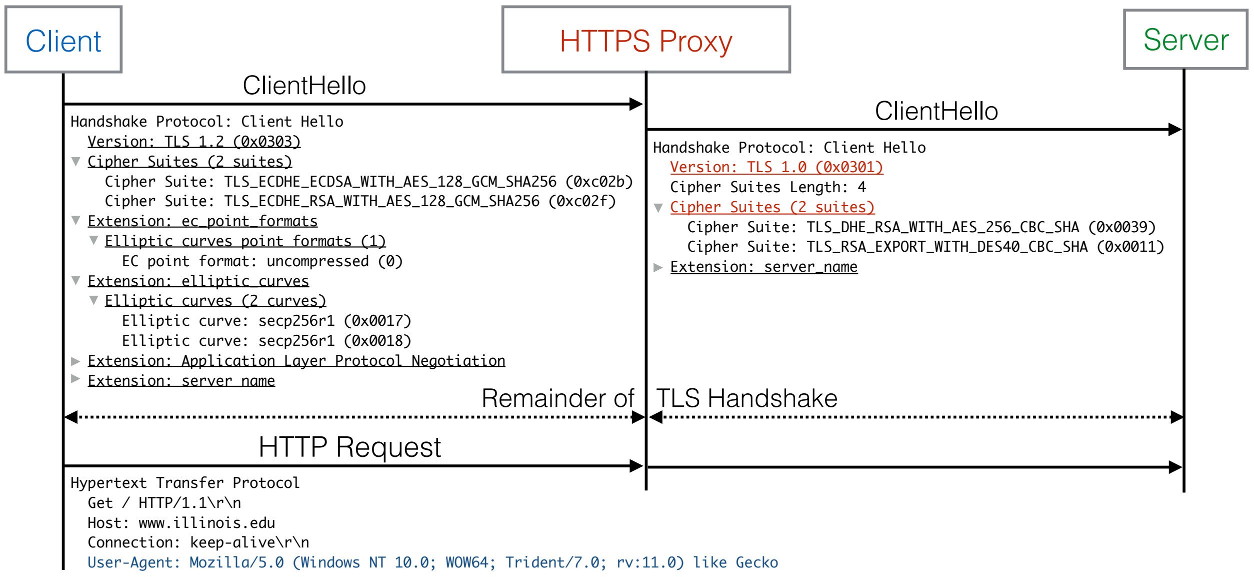
As of early 2016, there exist more than 340 cipher suites, 36 elliptic curves, 3 elliptic curve point formats, 28 signature algorithms, and 27 extensions that clients can advertise [156, 157]. In practice, browsers and security products use varying TLS libraries and advertise different handshake parameters. As we will show in Section 4.2, these characteristic variations allow us to uniquely identify individual TLS implementations based on their handshakes.

4.1.3 HTTP User-Agent Header

The HTTP protocol allows the client and server to pass additional information during a connection by including header fields in their messages. For example, the client can include the `Accept-Charset: utf-8` header to indicate that it expects content to be encoded in UTF-8. One standard client header is the *User-Agent* header, which indicates the client browser and operating system in a standardized format. There has been significant prior study on User-Agent header spoofing. These studies have largely found that end users do not spoof their own User-Agent header [158, 159, 160]. For example, Eckersley found that only 0.03% of connections with a Firefox User-Agent supported features unique to Internet Explorer, indicating spoofing [159]. Fingerprinting studies commonly trust the User-Agent string [161], and we follow suit in this work.

4.2 TLS IMPLEMENTATION HEURISTICS

Our methodology for identifying interception is based on detecting a *mismatch* between the browser specified in the HTTP User-Agent header and the cryptographic parameters advertised during the TLS handshake (Figure 4.1). In this section, we characterize the handshakes from popular browsers and develop heuristics that determine whether a TLS handshake is consistent with a given browser. We then go on to fingerprint the handshakes produced by popular security products in order to identify the products responsible for interception in the wild.



```
Handshake Protocol: Client Hello
  Version: TLS 1.2 (0x0303)
▼ Cipher Suites (2 suites)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
▼ Extension: ec_point_formats
  ▼ Elliptic curves point formats (1)
      EC point format: uncompressed (0)
▼ Extension: elliptic_curves
  ▼ Elliptic curves (2 curves)
      Elliptic curve: secp256r1 (0x0017)
      Elliptic curve: secp256r1 (0x0018)
▶ Extension: Application Layer Protocol Negotiation
▶ Extension: server_name
```

```
Handshake Protocol: Client Hello
  Version: TLS 1.0 (0x0301)
  Cipher Suites Length: 4
▼ Cipher Suites (2 suites)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0011)
▶ Extension: server_name
```

```
Hypertext Transfer Protocol
  Get / HTTP/1.1\r\n
  Host: www.illinois.edu
  Connection: keep-alive\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

Figure 4.1: **HTTPS Interception**—Products monitor HTTPS connections by acting as transparent proxies that terminate the browser TLS session, inspect content, and establish a new connection to the destination server. These proxies use different TLS libraries than popular browsers, which allows us to detect interception by identifying a mismatch between the HTTP User-Agent header and TLS client behavior.

4.2.1 Web Browsers

We captured the TLS handshakes generated by the four most popular browsers: Chrome, Safari, Internet Explorer, and Firefox [162]. To account for older versions, differing operating systems, and varying mobile hardware, we generated and captured handshakes in different environments using BrowserStack [163], a cloud service that provides developers with a variety of virtual machines for testing websites.[2]

We analyzed the non-ephemeral parameters advertised in the TLS handshakes, finding that each browser family selects a unique set of options, and that these options differ from those used by both common libraries (e.g., OpenSSL) and popular interception products.[3] However, while each browser, library, and security product produces a unique Client Hello message, the parameters selected by browsers are not statically defined. Instead, browsers alter their behavior based on hardware support, operating system updates, and user preferences.

Instead of generating all possible permutations, we analyzed *when* browsers select different parameters and developed a set of heuristics that determine whether a specific handshake *could have*

[2]We analyzed Chrome 34–50 and Firefox 3–46 on Windows XP, 7, 8, 8.1, and 10 and Mac OS X Snow Leopard, Lion, Mountain Lion, Mavericks, Yosemite, and El Capitan. We additionally captured handshakes from Apple Safari 5–9, Internet Explorer 6–11, Microsoft Edge, and the default Android browser on Android 4.0–5.0. In total, we collected 874 fingerprints.

[3]We compare the presence and order of cipher suites, extensions, compression methods, elliptic curves, signature algorithms, and elliptic curve point formats.

been generated by a given browser. For example, none of the four browsers have ever supported the TLS Heartbeat extension [164]. If a browser connection advertises its support, we know that the session was intercepted. On the other hand, despite the fact that all four browsers have default support for AES-based ciphers, the lack of these ciphers does not indicate interception, because browsers allow users to disable specific cipher suites. This methodology has the advantage of excluding false positives that arise from uncommon user configurations. However, it can yield false negatives if a proxy copies TLS parameters from the original client connection.

We describe the heuristics for each browser below:

- **Mozilla Firefox** Firefox was the most consistent of the four browsers, and by default, each version produces a nearly identical Client Hello message regardless of operating system and platform. All parameters, including TLS extensions, ciphers, elliptic curves, and compression methods are predetermined and hard-coded by the browser. Users can disable individual ciphers, but they cannot add new ciphers nor reorder them. To determine whether a Firefox session has been intercepted, we check for the presence and order of extensions, cipher suites, elliptic curves, EC point formats, and handshake compression methods. Mozilla maintains its own TLS implementation, Mozilla Network Security Services (NSS) [165]. NSS specifies extensions in a different order than the other TLS libraries we tested, which allows it to be easily distinguished from other implementations. The library is unlikely to be directly integrated into proxies because it is seldom used in server-side applications.

- **Google Chrome** Chrome was one of the most challenging browsers to fingerprint because its behavior is dependent on hardware support and operating system. For example, Chrome prioritizes ChaCha-20-based ciphers on Android devices that lack hardware AES acceleration [166] and Chrome on Windows XP does not advertise support for ECDSA keys at all. These optimizations result in several valid cipher and extension orderings for each version of Chrome, and furthermore, users can disable individual cipher suites. We check for ciphers and extensions that Chrome is known to not support, but do not check for the inclusion of specific ciphers or extensions, nor do we validate their order. When appropriate, we check the presence and order of elliptic curves, compression methods, and EC point formats. We note that because Chrome uses BoringSSL, an OpenSSL fork, connections have a similar structure to OpenSSL. However, Chrome supports considerably fewer options, including a subset of the default extensions, ciphers, and elliptic curves advertised by OpenSSL.

- **Microsoft Internet Explorer and Edge** Internet Explorer is less consistent than the other browsers for two reasons: (1) administrators can enable new ciphers, disable default ciphers, and arbitrarily reorder cipher suites through Windows Group Policy and registry changes, and (2) IE uses the Microsoft SChannel library, an OS facility which behaves differently depending on *both* Windows updates and browser version. Because of this additional flexibility, it is hard to rule out handshakes that include outdated ciphers, so we introduce a third categorization, *unlikely*, which we use to indicate configurations that are possible but improbable in practice (e.g., including an export-grade cipher suite). We note that minor OS updates alter TLS behavior, but are not indicated by the HTTP User-Agent header. We mark behavior consistent with any OS update as valid. SChannel connections can by uniquely identified because SChannel is the only TLS library we tested that includes the OCSP status request extension before the supported groups and EC point formats extensions.

- **Apple Safari** Apple Safari ships with its own TLS implementation, Apple Secure Transport, which does not allow user customization. The order of ciphers and extensions is enforced in code. While extension order does not vary, the NPN, ALPN, and OCSP stapling extensions are frequently excluded, and the desktop and mobile versions of Safari have different behavior. One unique aspect of Secure Transport is that it includes the TLS_EMPTY_RENEGOTIATION_INFO_SCSV cipher first, whereas the other libraries we investigated include the cipher last. Similar to Microsoft, Apple has changed TLS behavior in minor OS updates, which are not indicated in the HTTP User-Agent header. We allow for any of the updates when validating handshakes, and we check for the presence and ordering of ciphers, extensions, elliptic curves, and compression methods.

4.2.2 Fingerprinting Security Products

While our heuristics enable us to detect *when* interception is occurring, they do not indicate *what* intercepted the connection. To identify products used in the wild, we installed and fingerprinted the Client Hello messages generated by well-known corporate middleboxes (Figure 4.2), antivirus software (Figure 4.3), and other client-side software previously found to intercept connections (e.g., SuperFish [167]). In this section, we describe these products.

We generate a fingerprint for each product by hashing the non-ephemeral parameters advertised in the Client Hello message (i.e., version, ciphers, extensions, compression methods, elliptic curves, and signature methods). When we detect that a handshake is inconsistent with the indicated browser, we check for an exact match with any of the fingerprints we collected. This strategy has several potential weaknesses. First, multiple products could share a single fingerprint. This seems particularly likely given that developers are likely to use one of a small number of popular TLS libraries (e.g., OpenSSL). Second, if products allow customization, then fingerprints of the default configuration will not match these customized versions.

Product	Grade	Validates Certificates	Modern Ciphers	Advertises RC4	TLS Version	Grading Notes
A10 vThunder SSL Insight	F	✓	✓	Yes	1.2	Advertises export ciphers
Blue Coat ProxySG 6642	A*	✓	✓	No	1.2	Mirrors client ciphers
Barracuda 610Vx Web Filter	C	✓	✗	Yes	1.0	Vulnerable to Logjam attack
Checkpoint Threat Prevention	F	✓	✗	Yes	1.0	Allows expired certificates
Cisco IronPort Web Security	F	✓	✓	Yes	1.2	Advertises export ciphers
Forcepoint TRITON AP-WEB Cloud	C	✓	✓	No	1.2	Accepts RC4 ciphers
Fortinet FortiGate 5.4.0	C	✓	✓	No	1.2	Vulnerable to Logjam attack
Juniper SRX Forward SSL Proxy	C	✓	✗	Yes	1.2	Advertises RC4 ciphers
Microsoft Threat Mgmt.Gateway	F	✗	✗	Yes	SSLv2	No certificate validation
Sophos SSL Inspection	C	✓	✓	Yes	1.2	Advertises RC4 ciphers
Untangle NG Firewall	C	✓	✗	Yes	1.2	Advertises RC4 ciphers
WebTitan Gateway	F	✗	✓	Yes	1.2	Broken certificate validation

Figure 4.2: **Security of TLS Interception Middleboxes**—We evaluate popular network middleboxes that act as TLS interception proxies. We find that nearly all reduce connection security and five introduce severe vulnerabilities. *Mirrors browser ciphers.

Surprisingly, we find that none of the browsers nor the 104 products we manually investigated shared fingerprints, except for eight pieces of unwanted software that all use the Komodia Redirector SDK [168]. In other words, these fingerprints uniquely identify a single product. None of the client-side security products we tested allow users to customize TLS settings. However, many of the corporate middleboxes allow administrators to specify custom cipher suites. In this situation, we would be able to detect that interception is occurring, but not identify the responsible product.

Middleboxes and Corporate Proxies Nearly every major networking hardware manufacturer—including Barracuda, Blue Coat, Cisco, and Juniper—produces middleboxes that support "SSL Inspection". These devices allow organizations to intercept TLS traffic at their network border for analysis, content filtering, and malware detection. In March 2015, Dormann documented products from nearly 60 manufacturers that advertise this functionality [169]. We configured and fingerprinted twelve appliance demos from well-known manufacturers (e.g., Cisco and Juniper) and anecdotally popular companies (e.g., A10 and Forcepoint), per Figure 4.2. We note one conspicuous absence: ZScaler SSL Inspection. ZScaler provides a cloud-based SSL inspection service, but did not provide us with a trial or demo.

Antivirus Software We installed, tested, and fingerprinted popular antivirus products based on the software documented by de Carné de Carnavalet and Mannan [170], products previously found to be intercepting connections [171, 172], and a report of popular antivirus products [173]. Products from twelve vendors inject a new root certificate and actively intercept TLS connections.[4] We list the products that intercept connections in Figure 4.3.

Unwanted Software and Malware Motivated by recent reports of unwanted software intercepting TLS connections, we fingerprinted the Komodia SDK [168], which is used by Superfish, Qustodio, and several pieces of malware [167, 174], and the NetFilter SDK [175], which is used by PrivDog.

We tested products in January–March, 2016. We are publishing our browser heuristics and product fingerprints at https://github.com/zakird/tlsfingerprints.

4.3 MEASURING TLS INTERCEPTION

We measured global interception rates by deploying our heuristics at three network vantage points: Mozilla Firefox update servers, a set of popular e-commerce websites, and the Cloudflare content distribution network. We observed 7.75 billion TLS handshakes across the three networks. By deploying the heuristics on different networks, we avoid the bias inherent of any single vantage point. However, as we will discuss in the next section, we find varying amounts of interception and abuse on each network. Below, we describe each perspective in detail:

Firefox Update Servers Firefox browsers routinely check for software updates by retrieving an XML document from a central Mozilla server over HTTPS. This check uses Firefox's standard TLS library (Mozilla NSS) and occurs every 24 hours while the browser is running and on browser launch if the last update occurred more than 24 hours prior. We used Bro [176] to monitor connections to `aus5.mozilla.org`—the update server used by Firefox versions 43–48—between February 14–26, 2016. During this period, we observed 4.36 billion connections from 45K ASes and 243 of the 249 ISO-defined countries. Because we collected traffic using an on-path monitor instead of on the web server, we do not have access to the HTTP User-Agent header. However,

[4]We found that the following products did not intercept HTTPS: 360 Total, Ahnlabs V3 Internet Security, Avira AV 2016, Comodo Internet Security, F-Secure Safe, G DATA products, K7 Total Security, Malwarebytes, McAfee Internet Security, Microsoft Windows Defender, Norton Security, Panda Internet Security 2016, Security Symantec Endpoint Protection, Tencent PC Manager, Trend Micro Maximum Security 10, and Webroot SecureAnywhere.

Product	OS	Browser MITM				Grade	Validates Certificates	Modern Ciphers	TLS Version	Grading Notes
		IE	Chrome	Firefox	Safari					
Avast . . .										
AV 11	Win	●	○	○		A*	✓	✓	1.2	Mirrors client ciphers
AV 11.7	Mac		●	●	●	F	✓	✓	1.2	Advertises DES
AVG . . .										
Internet Security 2015–6	Win	●	●	○		C	✓	✓	1.2	Advertises RC4
Bitdefender . . .										
Internet Security 2016	Win	●	●	●		C	✓	○	1.2	RC4, 768-bit D-H
Total Security Plus 2016	Win	●	●	●		C	✓	○	1.2	RC4, 768-bit D-H
AV Plus 2015–16	Win	●	●	●		C	✓	○	1.2	RC4, 768-bit D-H
Bullguard . . .										
Internet Security 16	Win	●	●	●		A*	✓	✓	1.2	Mirrors client ciphers
Internet Security 15	Win	●	●	●		F	✓	✕	1.0	Advertises DES
CYBERsitter . . .										
CYBERsitter 11	Win	●	●	●		F	✕	✕	1.2	No cert. validation, DES
Dr. Web . . .										
Security Space 11	Win	●	●	●		C	✓	○	1.2	RC4, FREAK
Dr. Web 11 for OS X	Mac		●	●	●	F	✓	✕	1.0	Export ciphers, DES, RC2
ESET . . .										
NOD32 AV 9	Win	●	●	●		F	○	○	1.2	Broken cert. validation
Kaspersky . . .										
Internet Security 16	Win	●	●	●		C	✓	✓	1.2	CRIME vulnerability
Total Security 16	Win	●	●	●		C	✓	✓	1.2	CRIME vulnerability
Internet Security 16	Mac		●	●	●	C	✓	✓	1.2	768-bit D-H
KinderGate . . .										
Parental Control 3	Win	●	●	●		F	○	✕	1.0	Broken cert. validation
Net Nanny . . .										
Net Nanny 7	Win	●	●	●		F	✓	✓	1.2	Anonymous ciphers
Net Nanny 7	Mac		●	●	●	F	✓	✓	1.2	Anonymous ciphers
PC Pandora . . .										
PC Pandora 7	Win	●	◐	◐		F	✕	✕	1.0	No certificate validation
Qustodio . . .										
Parental Control 2015	Mac		●	●	●	F	✓	✓	1.2	Advertises DES

Interception:	**Certificate Validation:**	**Modern Ciphers:**
○ No Interception (conn. allowed)	✕ No Validation	✕ No Support
◐ Connections Blocked	○ Broken Validation	○ Non-preferred Support
● Connections Intercepted	✓ Correct Validation	✓ Preferred Support

Figure 4.3: **Security of Client-side Interception Software**—We evaluate and fingerprint popular antivirus and client-side security products, finding that products from twelve vendors intercept connections.[5] In all but two cases, products degrade TLS connection security. *Mirrors browser ciphers.

Vantage Point	% HTTPS Connections Intercepted		
	No Interception	Likely	Confirmed
Cloudflare	88.6%	0.5%	10.9%
Firefox	96.0%	0.0%	4.0%
E-commerce	92.9%	0.9%	6.2%

Figure 4.4: **Detecting Interception**—We quantify HTTPS interception at three major Internet services. We estimate that 5–10% of connections are intercepted.

only specific versions of Firefox are configured to connect to the server. Instead of looking for a mismatch with the HTTP User-Agent, we look for a mismatch between TLS handshake and any of the Firefox versions configured to connect to the server. There is no user-accessible content available on the site and there should be negligible other traffic. This vantage point provides one of the cleanest perspectives on clients affected by TLS interception. However, it only provides data for Firefox, one of the browsers believed to be least affected by client-side interception software [170].

Popular E-commerce Sites During two weeks in August and September 2015, a set of popular e-commerce sites hosted JavaScript that loaded an invisible pixel from an external server that recorded the raw TLS Client Hello, HTTP User-Agent string, and client HTTP headers. This perspective sees traffic from all browsers, but may suffer from falsified User-Agent headers. However, because the measurement required JavaScript execution, the dataset excludes simple page fetches. The sites have an international presence, but the connections we observe are likely skewed towards desktop users because the e-commerce provider has popular mobile applications. The dataset has the added benefit that it contains HTTP headers beyond User-Agent, which allow another avenue for detecting interception: looking for proxy related headers (e.g., `X-Forwarded-For` and `X-BlueCoat-Via`) and the modifications documented by Weaver et al. [177].

Cloudflare Cloudflare is a popular CDN and DDoS protection company that serves approximately 5% of *all* web traffic [178]. Cloudflare provides these services by acting as a reverse proxy. Clients connect to one of Cloudflare's servers when accessing a website, which serve cached content or proxy the connection to the origin web server. We logged the raw TLS Client Hello messages and HTTP User-Agent for a random 0.5% sample of all TLS connections to Cloudflare's frontend between May 13–20, 2016. We measure interception by detecting mismatches between the HTTP User-Agent and the TLS handshake. Cloudflare provides a more representative sample of browsers than the Firefox update servers. However, one of Cloudflare's foremost goals is to prevent DDoS attacks and other abuse (e.g., scripted login attempts), so the data is messier than the other two datasets, and a portion of connections likely have falsified HTTP User-Agent headers.

4.4 RESULTS

Our three vantage points provide varying perspectives on the total amount of interception: 4.0% of Firefox update connections, 6.2% of the e-commerce connections, and 10.9% of Cloudflare sessions in the United States (Figure 4.4). In all cases, this is more than an order of magnitude higher than previously estimated [171, 172].

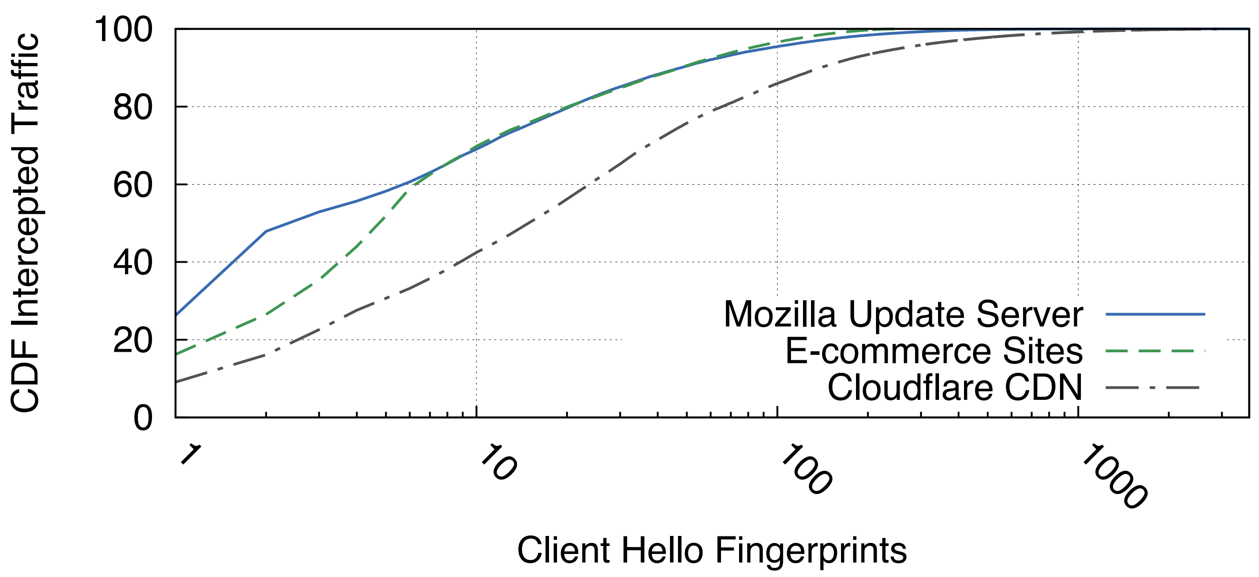


Figure 4.5: **CDF of Interception Fingerprints**—We fingerprint non-ephemeral parameters in replacement Client Hello messages to track the products that perform interception. Ten fingerprints account for 69% of the intercepted e-commerce traffic, 69% of Firefox update traffic, and 42% of Cloudflare traffic.

	Fingerprint Description	% Total
E-commerce	*Unknown*	17.1%
	Avast Antivirus	10.8%
	Unknown	9.4%
	Blue Coat	9.1%
	Unknown	8.3%
Cloudflare	Avast Antivirus	9.1%
	AVG Antivirus	7.0%
	Unknown. Likely AV; mainly Windows 10/Chrome 47	6.5%
	Kaspersky Antivirus	5.0%
	BitDefender Antivirus	3.1%
Firefox	Bouncy Castle (Android 5)	26.3%
	Bouncy Castle (Android 4)	21.6%
	Unknown. Predominantly India	5.0%
	ESET Antivirus	2.8%
	Dr. Web Antivirus	2.6%

Figure 4.6: **Top Interception Fingerprints—**We show the products responsible for the most interception at each vantage point.

Detection Method	Firefox	E-commerce	Cloudflare
Invalid Extensions	16.8%	85.6%	89.0%
Invalid Ciphers	98.1%	54.2%	68.7%
Invalid Version	–	2.0%	–
Invalid Curves	–	5.5%	9.4%
Invalid Extension Order	87.7%	33.9%	40.4%
Invalid Cipher Order	98.8%	21.2%	21.1%
Missing Required Ext.	97.9%	91.1%	50.9%
Injected HTTP Header	–	14.0%	–

Figure 4.7: **Handshake Mismatches**—We break down the mismatches used to detect intercepted sessions. For more than 85% of intercepted connections, we detect an invalid handshake based on the use of unsupported extensions, ciphers, or curves. Some features were unavailable for Firefox and Cloudflare.

4.4.1 Firefox Update Server

HTTPS connections for 4.0% of Firefox clients were intercepted, which is the lowest rate among the three perspectives. Interception is likely less common for Firefox users because the browser ships with its own certificate store, whereas Internet Explorer, Chrome, and Safari use the host operating system's root store. Prior work [170] and our own testing (Figure 4.3) both find that some antivirus products (e.g., Avast) will intercept connections from these other browsers but neglect to proxy Firefox sessions. In corporate environments, administrators can separately install additional root authorities in Firefox [179], but the added step may dissuade organizations that proxy connections from deploying the browser.

Sources of Interception The two most common interception fingerprints belong to the default configurations of Bouncy Castle on Android 4.x and 5.x, and account for 47% of intercepted clients (Figure 4.5). These fingerprints were concentrated in large ASes belonging to mobile wireless providers, including Verizon Wireless, AT&T, T-Mobile, and NTT Docomo (a Japanese mobile carrier). As can be seen in Figure 4.9, 35% of all Sprint and 25.5% of all Verizon Firefox connections (including non-intercepted) matched one of the two fingerprints. It is possible to intercept TLS connections on Android using the VPN and/or WiFi permissions. However, given the default values, it is unclear exactly which Android application is responsible for the interception. Bouncy Castle on Android 5.x provides reasonable ciphers equivalent to a modern browser; on Android 4.x, Bouncy Castle advertises export-grade cipher suites, making it vulnerable to interception by an on-path attacker. The third most common fingerprint accounts for 5.3% of Firefox traffic. We were not able to identify the product associated with the fingerprint but note that nearly half of its traffic occurred in India and its diurnal and weekend patterns are consistent with home antivirus or

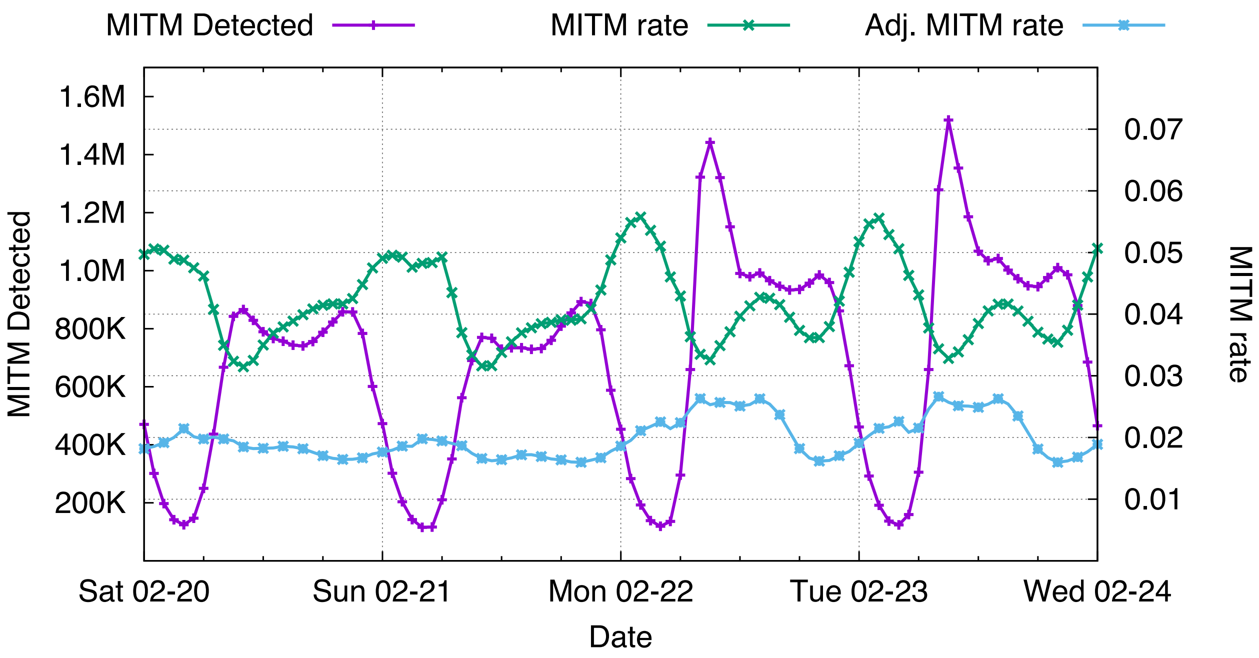


Figure 4.8: **Temporal Variation in Firefox Interception**—We observe the highest amount of raw interception during times of peak traffic, but the highest interception rates during periods of low total traffic. This is likely because the two largest fingerprints are associated with mobile carriers and will update at night when desktop computers are powered off. Excluding these two fingerprints, interception remains relatively stable.

malware.

Temporal Pattern The number of raw intercepted connections mirrors the diurnal pattern of all Firefox traffic. As can be seen in Figure 4.8, there are typically more connections on weekdays, and we observe the peak number of connections on weekday mornings. This intuitively aligns with the first computer access of the day triggering a connection to the Firefox update server. Oddly, though, the percentage of intercepted traffic is inversely proportional to total traffic, peaking near midnight and in the early morning. When we remove the two Android Bouncy Castle fingerprints, the total percentage of intercepted connections decreases by 47% and no longer peaks during these off hours. We suspect that the interception peaks are the result of mobile devices remaining on at night when other desktop computers are powered off. Weekend interception rates level out around 2% and increase during the weekdays, suggesting the presence of corporate TLS proxies.

Geographic Disparities Interception is more prevalent in several countries (Figure 4.10). For example, 15% of TLS connections from Guatemala were intercepted, a rate 3–4 times higher than the global average. This is primarily due to COMCEL, a mobile provider responsible for 34.6% of all Guatemalan update server traffic, having a 32.9% interception rate. Greenland has the second

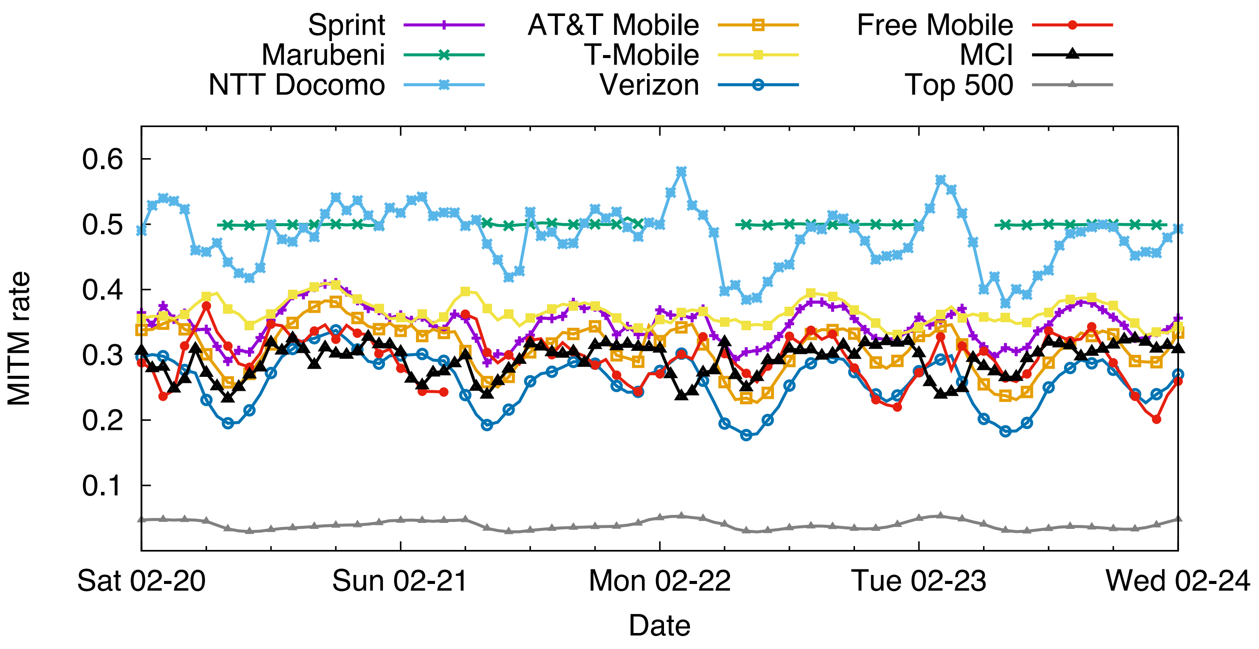


Figure 4.9: **ASes with Highest Firefox Interception**—We find that 8 ASes have significantly higher interception rates within the top 500 ASes. All but one are mobile providers.

Country	**MITM %**	**Country**	**MITM %**
Guatemala	15.0%	Kiribati	8.2%
Greenland	9.9%	Iran	8.1%
South Korea	8.8%	Tanzania	7.3%
Kuwait	8.5%	Bahrain	7.3%
Qatar	8.4%	Afghanistan	6.7%

Figure 4.10: **Countries with Highest Firefox Interception**—We show the ten countries with the highest interception rates when connecting to the Mozilla update server. Countries with above average interception rates generally have a large amount of traffic intercepted by a single, dominant mobile provider.

highest interception rate (9.9%), which is caused by a single AS: TELE Greenland. Nearly half of the interception is performed by a Fortigate middlebox. The third most commonly intercepted country, South Korea, is one of the most highly mobile-connected countries in the world [180]. In general, large ASes with above average interception belong to mobile providers and fluctuate between 20% to 55% interception depending on the time of day, as seen in Figure 4.9. The single exception is Marubeni OKI Network Solutions, which maintains a consistent interception rate around 50% that begins in the morning and ends near midnight everyday. It is unclear what behavior results in this temporal pattern.

4.4.2 Popular E-commerce Sites

The e-commerce dataset is composed of visits to set of popular e-commerce websites and is not limited to a specific browser version. To account for this, we parsed the HTTP User-Agent header and identified mismatches between the announced browser and TLS handshake. We observed 257K unique User-Agent headers, and successfully parsed the header for 99.5% of connections. The browsers we fingerprinted account for 96.1% of connections; 2.5% belong to browsers we did not fingerprint and 1.4% are from old browser versions.

We find that 6.8% of connections were intercepted and another 0.9% were likely intercepted, but cannot be definitively classified. For the connections where we could detect a specific fingerprint, 58% belong to antivirus software and 35% to corporate proxies. Only 1.0% of intercepted traffic is attributed to malware (e.g., SuperFish), and the remaining 6% belong to miscellaneous categories. The three most prevalent known fingerprints belong to Avast Antivirus, Blue Coat, and AVG Antivirus, which account for 10.8%, 9.1%, and 7.6% of intercepted connections, respectively.

The e-commerce dataset also includes HTTP headers, which allow us to identify a subset of connections that were intercepted by network middleboxes, but do not match any our existing fingerprints. We find proxy-related headers in 14.0% of invalid handshakes, most prominently `X-BlueCoat-Via`, `Via`, `X-Forwarded-For`, and `Client-IP`. We additionally use these headers to detect interception. We detected 96.1% of interception based on a version mismatch or the presence of invalid extensions or ciphers. Another 2.2% of intercepted connections lacked expected extensions, 0.7% used invalid cipher or extension ordering, and 1.6% contained proxy-related HTTP headers (Figure 4.7).

Chrome accounts for 40.3% of TLS traffic, of which 8.6% was intercepted, the highest interception rate of any browser. On the other extreme, only 0.9% of Mobile Safari connections were intercepted. Interception is far more prominent on Windows, where we see 8.3%–9.6% interception compared to 2.1% on Mac OS. This is likely because most corporate users use Windows in the workplace, and many antivirus products that perform interception are Windows-based. We summarize these results in Figure 4.13.

Falsified User-Agents It is possible that some connections in the e-commerce dataset have falsified User-Agents headers, which would artificially inflate the interception rate. We intuitively expect that handshakes belonging to interception products will have a larger number of associated User-Agents than a custom crawler with a falsified User-Agent. All but a handful of the TLS products we investigated have at least 50 associated User-Agents, and at the extreme, Avast Antivirus and Blue Coat's corporate proxy had 1.8K and 5.2K associated User-Agents, respectively. When we excluded interception fingerprints associated with less than 50 User-Agents, the global interception rate dropped from 6.8% to 6.2%. Given this modest decrease, we suspect that the mismatches we detect are due to interception instead of spoofed User-Agents. However, we take a conservative route and restrict our analysis to the 6.2%.

During our analysis, we noted two irregularities. First, nearly 3 million connections (approximately 75% of which had a Blue Coat header) used the generic User-Agent string "`Mozilla/4.0 (compatible;)`"). Cisco has documented that Blue Coat devices will frequently mask browser User-Agents with this generic agent string [181]. Despite knowing Blue Coat devices intercept these TLS connections, we take the most conservative approach and exclude these connections from our analysis because we do not know what percentage of devices behind the proxy would have been identifiable. However, we note that if we assume the same proportion of identifiable browsers as the general population (95.6%), Blue Coat would be the second largest fingerprint and the total percentage of connections intercepted would rise from 6.2% to 7.0%. Second, we find that over 90% of Internet Explorer connections on Windows XP appear intercepted because they include modern ciphers and extensions that were not previously documented on XP, nor that we could reproduce. We exclude these connections.

4.4.3 Cloudflare

The Cloudflare network provides perhaps the most representative view of global HTTPS traffic, but also the messiest. We initially observe a 31% interception rate, which is 3–7 times higher than the other vantage points. This elevated interception rate is likely due to abuse (e.g., login attempts or content scraping) and falsified User-Agent headers—some of the very types of requests that Cloudflare protects against. The Firefox server relied on only Firefox browsers accessing an obscure update server and the e-commerce websites required JavaScript execution in order to record a TLS connection. In contrast, the Cloudflare data reflects all TLS connections across a broad range of websites, so even simple command line utilities such as *wget* and *curl* can appear in the Cloudflare dataset with falsified User-Agent headers.

We take several steps to account for this abuse. First, we removed fingerprints associated with fewer than 50 unique User-Agent headers. Next, we limited our analysis to the 50 largest ASes that are not cloud or hosting providers. Unfortunately, even after this filter, we still observe an artificially high interception rate ranging from 11% in the Americas to 42% in Asia. We find that while large ASes in the U.S. have clear purposes, the majority of networks in Europe and Asia do not. In Asia, numbers varied widely and most ASes had little description. In Europe, ASes would frequently span multiple countries and contain requests that appeared to from both home users and hosting providers. We limit our analysis to the ASes from the top 50 that were located in the United States and primarily serve end users. While this reduces the scope of the dataset, there are

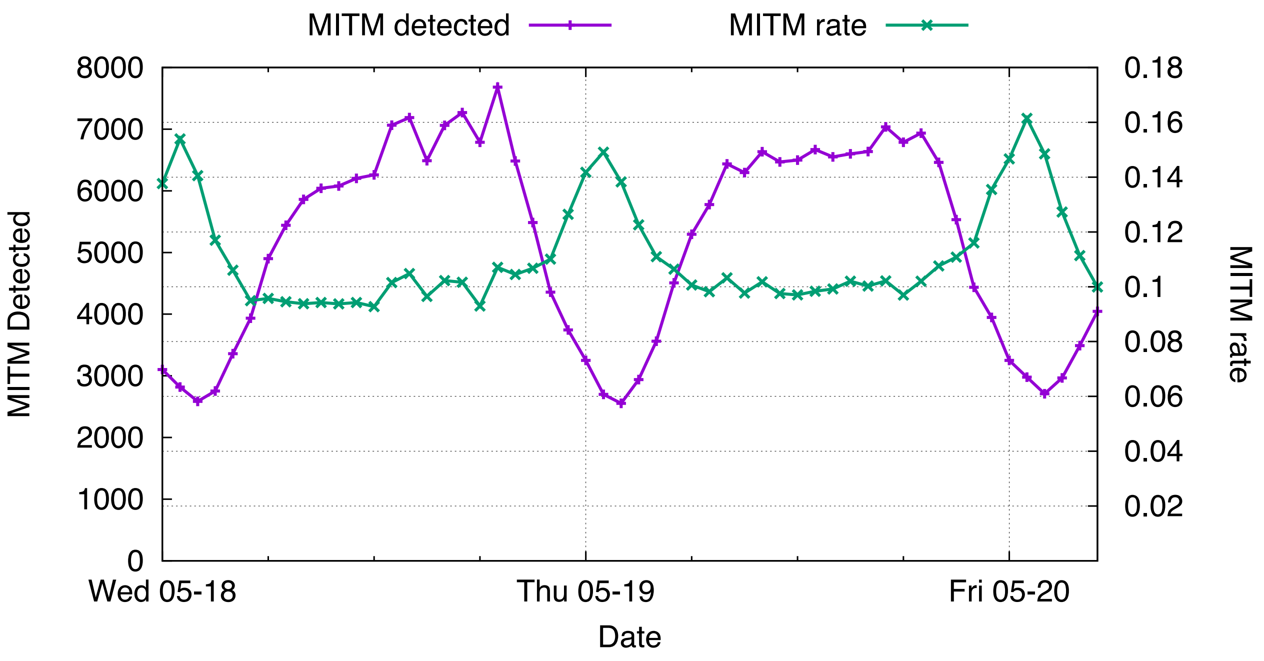


Figure 4.11: **Temporal Variation in Cloudflare Interception**— We observe the highest amount of raw interception during times of peak traffic, but the highest interception rates during periods of low total traffic. This is likely due to higher percentages of automated bot traffic where the User-Agent header is spoofed.

Network Type	No Interception	Likely	Confirmed
Residential/Business	86.0%	0.4%	13.6%
Cell Provider	94.1%	0.1%	5.8%

Figure 4.12: **U.S. Network Breakdown**—We show the Cloudflare interception rates for types of U.S. networks.

lower interception rates in the U.S. compared to any other region, providing a conservative lower bound.

In the U.S., we observe a 10.9% interception rate, with a stark contrast between mobile ASes (5.2–6.5%) and residential/enterprise ASes 10.3–16.9%), per Figure 4.12. Four of the top five handshake fingerprints belong to antivirus providers: Avast, AVG, Kaspersky, and BitDefender, which are also prominent on the e-commerce sites. The remaining unidentified fingerprint primarily occurs for Chrome 47 on Windows 10. Despite the alignment with a specific browser version and OS—which might indicate an incorrect heuristic—we confirm that this handshake cannot be produced by Chrome and advertises 80 cipher suites including IDEA/CAMELLIA, diverging significantly from the Chrome family. The fingerprint also occurs consistently across non-mobile ASes and peaks usage during evening hours, suggesting malware or antivirus software. These five fingerprints account for 31% of intercepted traffic (Table 4.5).

E-commerce Sites			
Browser	All Traffic	Intercepted	Of Intercepted
Chrome	40.3%	8.6%	56.2%
Explorer	16.8%	7.4%	19.6%
Firefox	13.5%	8.4%	18.2%
Safari	10.2%	2.1%	3.4%
Chromium	7.6%	0.1%	0.1%
Mobile Safari	7.6%	0.9%	1.1%
Other	4.0%	4.0%	2.4%
OS	All Traffic	Intercepted	Of Intercepted
Windows 7	23.3%	9.6%	56.6%
Windows 10	22.5%	9.3%	14.3%
iOS	17.3%	0.1%	1.1%
Mac OS	15.8%	2.1%	6.5%
Android	9.4%	1.0%	0.5%
Windows 8.1	6.9%	8.3%	15.8%
Other	4.8%	21.4%	15.2%
Cloudflare			
Browser	All Traffic	Intercepted	Of Intercepted
Chrome	36.2%	14.7%	48.8%
Mobile Safari	17.5%	1.9%	3.3%
Explorer	14.9%	15.6%	21.2%
Safari	8.9%	6.5%	5.3%
Firefox	8.5%	18.2%	14.2%
Mobile Chrome	8.4%	4.7%	3.6%
Other	5.6%	7.0%	3.6%
OS	All Traffic	Intercepted	Of Intercepted
Windows 7	23.9%	13.4%	29.2%
Windows 10	22.9%	13.1%	27.4%
iOS	17.5%	2.0%	3.2%
Mac OS	16.0%	6.6%	9.6%
Android	9.5%	4.8%	4.2%
Windows 8.1	4.9%	24.4%	11.0%
Other	5.3%	31.7%	15.4%

Figure 4.13: **OS and Browser Breakdown**—We show the breakdown of all traffic, the amount of traffic intercepted, and percentage of all interception that each browser and operating system accounts for across both the e-commerce and Cloudflare vantage points.

Similar to Firefox updates, the total amount of HTTPS interception correlates with total HTTPS traffic, peaking in the middle of the day and declining during evening hours, but with the highest interception rates at night (Figure 4.11). This might be due to mobile traffic as we saw for Firefox but could also indicate the presence of bot traffic.

4.4.4 Results Summary and Validation

We can partially validate our methodology by checking whether we failed to detect any connections that included proxy-related HTTP headers as intercepted. We find that 1.6% of the e-commerce connections included proxy headers, but did not have evidence of interception in their TLS handshakes. This suggests that the methodology catches the vast majority of interception, but it does miss some edge cases.

To verify that our heuristics aren't incorrectly classifying valid handshakes, we investigated why our heuristics marked connections as intercepted. We detected more than 85% of intercepted connections based on the presence of known unsupported ciphers or extensions, rather than a missing extension or invalid ordering. More than 98% of intercepted connections in the Firefox dataset were found based on the inclusion of ciphers that have never been implemented in NSS, and 82% of all intercepted connections indicated support for the heartbeat extension—an immediate giveaway given that no browsers support the extension. This suggests that our heuristics are finding handshakes produced by other libraries rather than misclassifying browser edge cases.

However, while the methodology appears sound, the three perspectives we studied provide differing numbers on the total amount of interception. All three perspectives find more than an order of magnitude more interception than previously estimated, and we estimate that 5–10% of connections are intercepted. However, we offer a word of caution on the exact numbers, particularly for the Cloudflare dataset, where abuse may inflate the interception rate we observe.

4.5 IMPACT ON SECURITY

In this section, we investigate the security impact of HTTPS interception. First, we introduce a grading scale for quantifying TLS client security. Then, we investigate common interception products, evaluating the security of their TLS implementations. Based on these ratings and the features advertised in Client Hello messages, we quantify the change in security for intercepted connections.

4.5.1 Client Security Grading Scale

There does not exist a standardized rubric for rating TLS client security. We define and use the following scale to consistently rate browsers, interception products, and the connections we observe in the wild:

A: Optimal. The TLS connection is equivalent to a modern web browser in terms of both security and performance. When grading cipher suites, we specifically use Chrome's definition of "secure TLS" [182].

B: Suboptimal. The connection uses non-ideal settings (e.g., non-PFS ciphers), but is not vulnerable to known attacks.

C: Known Attack. The connection is vulnerable to known TLS attacks (e.g., BEAST, FREAK, and Logjam), accepts 768-bit Diffie-Hellman parameters, or advertises RC4support.

F: Severely Broken. The connection is severely broken such that an active man-in-the-middle attacker could intercept and decrypt the session. For example, the product does not validate certificates, or offers known-broken cipher suites (e.g., DES).

Our grading scale focuses on the security of the TLS handshake and does not account for the additional HTTPS validation checks present in many browsers, such as HSTS, HPKP, OneCRL/CRLSets, certificate transparency validation, and OCSP must-staple. None of the products we tested supported these features. Therefore, products that receive an A grade for their TLS security likely still reduce overall security when compared to recent versions of Chrome or Firefox.

4.5.2 Testing Security Products

To measure the security of the connections initiated by interception products, we installed the trial versions of the corporate proxies, popular client security software, and malware listed in Section 4.2.[5] We then ran the latest version of Chrome, Internet Explorer, Firefox, and Safari through each product, visiting a website that executed the following tests:

1. **TLS Version.** We check the highest version of TLS that the product supports. We grade any client that supports at best TLS 1.1 as B, SSLv3 as C, and SSLv2 as F.

2. **Cipher Suites.** We investigate the cipher suites present in the Client Hello. We rate any product that does not support Chrome's Strong TLS ciphers [182] as B, handshakes that

[5]Product demos could have different security profiles than their production counterparts. However, during our disclosure process, none of the manufacturers we contacted indicated this.

offer RC4 as C, and any product that advertises broken ciphers (e.g., export-grade or DES) as F.

3. **Certificate Validation.** We present a series of untrusted certificates, including expired, self-signed, and invalidly-signed certificates. We further test with certificates signed by CAs with a publicly-known private key (i.e., Superfish [183], eDell, and Dell provider roots [184]). We rate any product that accepts one of these certificates as F.

4. **Known TLS Attacks.** We check whether clients are vulnerable to the BEAST, FREAK, Heartbleed, and Logjam attacks, or accept weak Diffie-Hellman keys. We rate any vulnerable client as C.[6]

Corporate Middleboxes The default configurations for eleven of the twelve middleboxes we tested weaken connection security, and five of the twelve products introduce severe vulnerabilities that would enable future interception by a man-in-the-middle attacker. Ten support RC4-based ciphers, two advertise export-grade ciphers, and three have broken certificate validation (Figure 4.2). We note that the installation process for many of these proxies is convoluted, crash-prone, and at times, non-deterministic. Configuration is equally confusing, oftentimes with little to no documentation. For example, Cisco devices allow administrators to customize permitted cipher suites, but do not provide a list of ciphers to choose from. Instead, the device provides an undocumented text box that appears to accept OpenSSL cipher rules (e.g., `ALL:!ADH:@STRENGTH`). We suspect that this poor usability contributes to the abysmal configurations we see in the wild.

We were not able to acquire a demo or trial of the ZScaler Proxy, a prominently advertised cloud-based middlebox. Instead, we investigated the traffic that originated from one of the seven ZScaler ASes in the Cloudflare dataset.[7] We found one predominant handshake fingerprint that accounts for more than four times more traffic than the second most popular, and does not match any popular browsers indicated in the associated User-Agent strings. We would have rated this handshake at best B due to the lack of any perfect-forward-secret ciphers. However, we exclude it from any other analysis, because we were not able to check for further vulnerabilities.

Client-side Security Software In line with de Carné de Carnavalet and Mannan [170], we find that nearly all of the client-side security products we tested reduce connection security and ten introduce severe vulnerabilities (Figure 4.3). We note that these security grades are a lower bound

[6]We tested these products in January–March, 2016, which was approximately eight months after the Logjam disclosure and eleven months after FREAK.

[7]We investigated ASes 62907, 55242, 53813, 53444, 40384, 32921, and 22616.

that assume TLS stacks have no additional vulnerabilities. However, in practice, researchers discover bugs in antivirus software regularly. For Avast alone, ten vulnerabilities have been publicly disclosed within the last eight months, one of which allowed remote code execution through carefully crafted certificates [185].

Malware and Unwanted Software Researchers have previously found that Komodia does not validate certificates [183] and we find that the NetFilter SDK similarly does not properly validate certificate chains. We grade both as F: severely broken.

4.5.3 Impact on TLS Traffic

While many security products have insecure defaults, intercepted connections could have a different security profile. Security might be improved if administrators configure middleboxes to perform responsible handshakes, or, even with their poor security, proxies might protect further out-of-date clients. We investigated the security of intercepted handshakes based on the parameters advertised in the handshake (e.g., TLS version and cipher suites), and in the cases where we can identify the interception product, its security rating. To determine the change in security rather than just the security of the new connection, we calculated the security of the browser version specified in the HTTP User-Agent and compare that to the security of the handshake we observe.

Similar to how each of our three networks has a different interception rate, each vantage point presents a different security impact. For Firefox, 65% of intercepted connections have reduced security and an astounding 37% have negligent security vulnerabilities. 27% of the e-commerce and 45% of the Cloudflare connections have reduced security, and 18% and 16% are vulnerable to interception, respectively.

Interception products increased the security for 4% of the e-commerce and 14% of the Cloudflare connections. The discrepancy in increased security is largely due to temporal differences in data collection and the deprecation of RC4 cipher suites during this period. When the e-commerce sites collected data in August 2015, browsers considered RC4 to be safe. However, between August 2015 and April 2016, standards bodies began advising against RC4 [186] and both Chrome and Firebox deprecated the cipher [187]. When grading Cloudflare connections in May 2016, we labeled connections that advertised RC4 as C. This results in connections from older versions of Internet Explorer and Safari being marked insecure, and proxies improving the security for an increased number of connections.

Corporate Middleboxes During our earlier analysis of corporate proxies, we found that many network middleboxes inject HTTP headers, such as `X-Forwarded-For` and `Via`, to assist managing simultaneous proxied connections. We analyzed the connections in the e-commerce dataset with proxy-related headers to better understand the security of corporate middleboxes compared to client-side software. We find that connection security is significantly worse for middleboxes than the general case. As can be seen in Figures 4.14 and 4.15, security is degraded for 62.3% of connections that traverse a middlebox and an astounding 58.1% of connections have severe vulnerabilities.

We note a similar phenomenon in the Firefox data where we manually investigated the top 25 ASes with more than 100K connections, the highest interception rates, and a single predominant interception fingerprint. We primarily find financial firms, government agencies, and educational institutions. With the exception of one bank, 24 of the top 25 ASes have worsened security due to interception. For 12 of the 25 ASes, the predominant TLS handshake includes export-grade cipher suites, making them vulnerable to future interception by an active man-in-the-middle attacker.

4.6 DISCUSSION

While the security community has long known that security products intercept TLS connections, we have largely ignored the issue. We find that interception is occurring more pervasively than previously estimated and in many cases, introduces significant vulnerabilities. In this section, we discuss the implications of our measurements and make recommendations for both vendors and the security community.

We need community consensus. There is little consensus within the security community on whether HTTPS interception is acceptable. On the one hand, Chrome and Firefox have provided tacit approval by allowing locally installed roots to bypass key pinning restrictions [154]. However, at the same time, discussions over protocol features that facilitate safer interception have been met with great hostility within standards groups [188, 189]. These communities need to reach consensus on whether interception is appropriate in order to develop sustainable, long-term solutions.

We should reconsider where validation occurs. Many HTTPS security features expect connections to be end-to-end by mixing the HTTP and TLS layers, and by implementing HTTPS features in browser code rather than in TLS libraries. For example, to overcome weaknesses in existing revocation protocols, Firefox ships with OneCRL [190] and Chrome, CRLSets [191]. Both of these solutions increase browser security in the typical end-to-end case. However, these solutions provide no protection in the presence of a TLS proxy and because the solution is not part of the TLS protocol itself, TLS libraries do not implement these safe revocation checks. In a second example, HPKP directives are passed over HTTP rather than during the TLS handshake. Browsers cannot perform HPKP validation for proxied connections because they do not have access to the destination certificate and proxies do not perform this validation in practice.

While it is possible for proxies to perform this additional verification, they are not doing so, and in many cases vendors are struggling to correctly deploy existing TLS libraries, let alone implement additional security features. Given this evidence, our community needs to decide what roles should be carried out by the browser versus TLS implementation. If we expect browsers to perform this additional verification, proxies need a mechanism to pass connection details (i.e., server certificate and cryptographic parameters) to the browser. If we expect proxies to perform this validation, we need to standardize these validation steps in TLS and implement them in popular libraries. Unfortunately the current situation, in which we ignore proxy behavior, results in the worst case scenario where neither party is performing strict validation.

Cryptographic libraries need secure defaults. Several proxies deployed TLS libraries with minimal customization. Unfortunately the default settings for these libraries were vulnerable rendering the middlebox vulnerable. Client libraries and web servers need to prioritize making their products safe by default. We applaud OpenSSL's recent decision to remove known-broken cipher suites [192]. However, this change should have occurred more than a decade earlier and libraries continue to accept other weak options. Our community should continue restricting default options to known safe configurations.

Antivirus vendors should reconsider intercepting HTTPS. Antivirus software operates locally and already has access to the local filesystem, browser memory, and any content loaded over HTTPS. Given their history of both TLS misconfigurations [170] and RCE vulnerabilities [185], we strongly encourage antivirus providers to reconsider whether intercepting HTTPS is responsible.

Security companies are acting negligently. Many of the vulnerabilities we find in antivirus products and corporate middleboxes—such as failing to validate certificates and advertising broken ciphers—are negligent and another data point in a worrying trend of security products worsening security rather than improving it [170, 193]. We hope that by disclosing vulnerabilities in existing products, we can encourage manufacturers to patch problems. We urge companies to prioritize the security of their TLS implementations and to consider the pace at which the HTTPS ecosystem evolves and whether they can keep up with the necessary updates.

Do not rely on client configuration. Because cryptographic parameters must be supported by both the client and server, the security community has largely ignored HTTPS servers' lenient cipher support with the implied understanding that browser vendors will only advertise secure parameters. In 2015, Durumeric et al. found that nearly 37% of browser-trusted HTTPS servers on the IPv4 address space supported RSA export ciphers [194] despite their known weaknesses and discontinued use. It was only after the discovery of the FREAK attack—a bug in OpenSSL that allowed an active attacker to downgrade connections to export-grade cryptography—that operators began to actively disable export cipher suites.

While modern browsers are not vulnerable to active downgrade attacks, nearly two thirds of connections that traverse a network middlebox advertise export ciphers and nearly 3% of *all* HTTPS connections to the e-commerce sites included at least one export-grade cipher suite. While these products would optimally not be vulnerable, their risk can be reduced by encouraging websites to disable weak ciphers. Similarly, some interception products support secure ciphers, but do not order them correctly. In this situation, servers that explicitly choose strong ciphers will negotiate a more secure connection than those that honor client preference. We need to practice defense-in-depth and encourage both clients and servers to select secure parameters instead of relying on one side to always act sanely.

Administrators need to test middleboxes. Many of the products we tested support more secure connections with additional configuration. Unfortunately, this does not appear to be happening in practice. While manufacturers clearly need to improve their default settings, we also encourage the administrators who are deploying proxies to test their configurations using sites such as Qualys SSL Lab's client test, https://howsmyssl.com, and https://badssl.com. To incentivize better software, servers could consider checking client configuration and rejecting insecure clients.

4.7 SUMMARY

In this chapter, we detected 4–10% global TLS interception through a novel technique that constrasts the self-declared identity of a HTTPS User-Agent with the TLS software detected through fingerprinting. This interception indicates that a large number of systems contain additional trust anchors in their root stores that are injected by antivirus or middlebox software. We find that these injected trust anchors are operated by largely insecure software that expose users to critical risk in over 50% of the products analyzed. One potential solution to this issue is to devise explicit TLS interception methods that do not need to hijack existing root store trust.

Network	Increased Security	Decreased Security	Severely Broken
E-commerce (All Traffic)	4.1%	26.5%	17.7%
E-commerce (Middleboxes)	0.9%	62.3%	58.1%
Cloudflare	14.0%	45.3%	16.0%
Firefox Updates	0.0%	65.7%	36.8%

Figure 4.14: **Impact of Interception—**We summarize the security impact of HTTPS interception, comparing client–proxy connection security with proxy–server connection security.

Dataset	Original Security	New Security			
		→A	→B	→C	→F
Firefox	A→	34.3%	16.8%	12.2%	36.8%
E-commerce Sites: All Traffic	A→	57.1%	2.9%	5.6%	8.1%
	B→	2.7%	10.2%	1.2%	8.3%
	C→	0.6%	0.4%	1.0%	0.3%
	F→	0.0%	0.2%	0.1%	1.0%
E-commerce Sites: Middleboxes	A→	13.5%	3.0%	0.8%	18.0%
	B→	0.7%	23.3%	0.6%	37.8%
	C→	0.1%	0.1%	0.0%	2.2%
	F→	0.0%	0.0%	0.0%	0.0%
Cloudflare	A→	17.3%	1.1%	29.7%	10.0%
	B→	0.0%	0.0%	0.0%	0.0%
	C→	9.4%	3.3%	22.0%	4.5%
	F→	0.8%	0.1%	0.4%	1.5%

Figure 4.15: **Change in Security—**We calculate the change in connection security based on the parameters advertised in the Client Hello message and the security of the browser in the HTTP User-Agent header.

CHAPTER 5: CONCLUSION

Trust management in the web PKI is understudied due to its opacity, which manifests as data unavailability and poorly specified processes. In this thesis we increased the transparency of trust management by discovering the origins of root stores and individual trust anchors. We first bridged the gap between trusted entities (i.e., CAs) and trust instruments (i.e., X.509 CA certificates) to better identify who is trusted when a given trust anchor is included in a root store. We then considered a broad range of different TLS clients, libraries, and operating systems, and found that their trust all bottlenecks at three distinct root programs. Finally, we devised a new technique to detect entities that have injected additional trust anchors into default root stores. Not only do we increase the transparency of the trust management ecosystem, we also shed light on security concerns that would benefit from explicit and transparent mechanisms to link trust anchors to their source.

5.1 INSIGHTS

From our findings, we derive a set of broader insights into the web PKI and trust-based delegated authentication.

Externalities of trust extensibility. The extensibility of trust—TLS ClientHello configurability, TLS extensions, X.509 extensions—is a key design property of the web PKI. It allows expansion of trust purposes (e.g., code signing), adaptability to exploits (e.g., renegotiation defense [195]), and addition of new features (e.g., OCSP must-staple [196]). However, as shown by this work, extensibility is not without its side effects: fingerprinting, fracturing, and misconfiguration.

– *Fingerprinting.* We take advantage of extensibility in both X.509/ASN.1 and TLS to develop fingerprinting techniques that provide new perspectives on previously opaque aspects of certificate issuance and TLS interception. In the absence of explicit transparency, these fingerprinting methods make this thesis possible, and provide a foundation for future transparency efforts.

– *Fracturing.* We find that extensibility in root store trust leads to fracturing: NSS derivatives currently do not support the more nuanced trust adopted by NSS, thereby partitioning the ecosystem into first- and second-class root stores. New update and explicit provenance methods can help re-unify the major root store providers.

- *Misconfiguration.* Looking at TLS interception products, we detect extensive, poorly configured usage of old and broken cryptographic primitives, which also enable many TLS attacks (e.g., Logjam, BEAST, POODLE, etc.). Operator notification and accelerated deprecation research can help mitigate these attractive attack targets.

By identifying these consequences of extensibility, we can begin to address the negative externalities while benefiting further from fingerprinting. Similarly, when building new secure systems, efforts must be taken to appropriately tune the benefits of extensibility and lessen negative externalities.

Looking behind the digital identity veil. As we have seen with root store trust in CAs, digital identity (i.e., X.509 certificates) is often a proxy for some real-world identity that we ultimately care about. The intention of trust in a root certificate is to trust the CA that operates the certificate. Unfortunately, this proxying can be fickle. Similar to a physical key with someone's name written on it, digital identity is easily and infinitely transferable, and we find that CAs do acquire and sell their root keying material. Today's cryptographic systems often equate identity with the ability to open a metaphorical door that is locked by a named key. Thus, it is critical to not only assess the real-world owners of digital identities, but also to constantly re-assess them. Without this assessment, the term "digital identity" is a misnomer as it provides no information about the real world operator.

Good trust is hard; imitation is tricky. Existing real-world identity verification techniques (e.g., annual audits of CAs and their keys) are far from foolproof and do not scale well, so today's web PKI must rely at least partially on trust. At a minimum, root store programs trust that CAs 1) will not give their digital identities (i.e., keying material) to other entities, 2) will sufficiently safeguard their digital identities from theft/exposure, and 3) will appropriately alert root programs in the case of any incidents. In order to make these highly-consequential trust decisions, root store programs perform extensive due diligence through audit examination, CA interrogation, issuance hygiene analysis, and more. Given the difficulty of judicious root trust, we find that many root store providers forgo this process and instead copy their root store from others. However, as shown by our root store ecosystem analysis, this shortcut to trust management is not as simple as it may seem and can often lead to outdated or risky root store trust. Future work should focus on two thrusts: simplifying/objectifying CA trust decisions and facilitating correct copying of foundational root stores.

Secure trust requires transparency. Trust is a critical element of web PKI authentication that allows it to scale, and in order to make good trust decisions, the trusting entities need to have transparency into who they are trusting. This applies not only for root store programs trusting CAs, but also for root store providers that decide to trust other root store programs, or TLS software developers/users deciding what root store to trust. Transparency in the public web PKI has few privacy concerns, and the main cost is maintenance burden that can likely be automated. This work has shown that knowledge of CA certificate operators has already influenced root store trust. Similar transparency efforts (e.g., connecting root stores to their root program and detailing the operators of TLS interception roots) will likely have a positive influence on trust management and improve overall web security.

5.2 FUTURE WORK

This thesis breaks through the first level of opacity in the web PKI and highlights the importance of transparency in further securing today's web connections. Hopefully our initial efforts catalyze exploration of trust management, which has been largely neglected in prior work. Our findings suggest several natural extensions for future research, discussed below.

5.2.1 Extending web PKI transparency

We believe that trust management in the web PKI can benefit from even greater transparency. Unlike other domains with privacy concerns for individual users or sensitive corporate data, PKI transparency has very few drawbacks as it deals with public entities (CAs and root store programs) that are securing the public web. Furthermore, in today's system, root store programs make CA trust decisions on behalf of users, who can benefit from trust management transparency. The risks of imprudent trust management are directly borne by users, and only indirectly affect root program operators through user attrition. Even technically minded users or system administrators have little insight into the decisions (and their impact) made by root program operators. This indirection is another layer of trust: users must trust the root programs that decide which CAs to trust. As such, just as CT helped to better inform root programs about CA behavior, trust management transparency will empower end users to better evaluate root store programs and make better trust decisions. Some future projects include:

- **Towards codifying trust policy and automating trust management.** The work described in this thesis sheds light on the behaviour of root store programs, finding differences in trust judgment between major players. The next step in transparency is to understand the policies that lead to the observed behavior. Unfortunately, root store policies can be subjective, underspecified, or inconsistent, which hampers both policy adherence and adherence testing. Codifying these policies (to the extent possible, since there may always be subjective terms) opens up an cornucopia of opportunities: linking specific policy items to trust/distrust incidents, comparing policy across root stores, automated policy checking, etc. These opportunities combined with measurements of CA operation, such as those developed in this thesis, will allow for new automated trust management methods.

- **Detecting the long tail of TLS authentication trust.** As the number of internet-connected devices continues to balloon, the long tail of TLS user agents (e.g., IoT devices) and dependence on the web PKI will likewise grow. Given the challenges of operating a root store program, and the inability of large root store providers (e.g., Linux distributions) to correctly copy established programs, we need to better understand the use cases for these smaller TLS deployments and make sure they are making secure trust management decisions. This work will develop new measurement techniques because the open-source collection of root stores for popular software does not scale well to the long tail of TLS user agents. One lower-bound approach is passive network monitoring to infer the certificates sent to TLS clients and detection of successful certificate validation. Another possibility for HTTPS user agents is the creation of *CA traversal sites* that cause a visitor to attempt a TLS connection with a handful of servers using certificates issued by different CAs. To do so ethically, we will use ecosystem-wide datasets to minimize any impact on individual web hosts.

5.2.2 PKI system design

PKIs are not unique to the web because many network technologies require the global authentication that PKI provides. At its core, PKI is a trust mechanism that enables scalable systems. By first exploring the web PKI, which is the most widely deployed PKI, we hope to develop a general systems approach to improving all PKIs.

- **A history of the web PKI: intentional design or runaway accident?** This systematization of knowledge paper (SoK) takes a retrospective look at the origin, development, and evolution of the web PKI, beginning with the specification of X.509 in the late 1980s. By tracking this intertwined ecosystem of standards, commercial CAs, root stores, policies, and legal frameworks, we will dissect the driving forces that have molded the web PKI into its current form. We will also identify the major failures and forks in the web PKI, eventually converging on a framework of PKI principles. This framework will enable evaluation of the strengths and weakness of today's PKI, while also suggesting promising alternative PKI designs.

- **PKI in the sky: a comparative evaluation of PKI deployments.** PKI systems are essential for nearly all global authentication schemes, and many have been proposed, but very few PKI deployments have achieved substantial and secure adoption. In this proposed SoK, we use the web PKI (the most widely deployed / successful PKI) as a measuring stick to assess other PKIs, including web of trust, RPKI, DNSSEC, code signing, email signature PKI, and DNS as a PKI. Our analysis will explore the deployment trajectories and gaps between web PKI and other PKIs, revealing the roadblocks to increased adoption and, ultimately, facilitating more secure networks.

- **A universal schema for contextual trust sharing.** After gaining a comprehensive understanding of the interplay between stakeholders in the web PKI and other global PKIs, we hope to develop a universal schema for trust. This new schema must be contextual, shareable, transition-friendly, and extensible.

  - *Contextual.* This thesis examines the trust anchors trusted by a variety of different systems, libraries, and clients. However, this understanding of trust is incomplete, as supplementary trust constraints are sometimes embedded in code due to the need for additional run-time context. Additionally, implementation quirks or bugs can accidentally constrict or expand the roots that are ultimately trusted. To better characterize the dynamic requirements of the web PKI, we will develop program analysis techniques to capture the full trust pipeline starting from certificate acquisition to a final trust decision.
  - *Shareable.* One of the major findings of this thesis is that web PKI trust is highly conserved: the overlap of trusted CAs between different root providers is relatively narrow, and root stores are often copied. This trust sharing occurs naturally in human society: individuals often adopt the trust decisions of their friends and family. A new trust schema must support explicit sharing and trust provenance.
  - *Transition-friendly.* As demonstrated by the sluggish adoption of non-web PKIs and other network protocol transitions (e.g., IPv6), transitioning existing network technologies is very difficult. A new trust schema must have transition compatibility as a primary design principle.
  - *Extensible.* In order to support a variety of protocols, including those that have not yet been conceived of, this new trust schema must support extension to different trust purposes (e.g., server authentication, code signing, etc.) and contexts (e.g., patching exploits, referred trust).

APPENDIX A: HISTORIC DISCLOSURE ISSUES

Issue #	Date	# Certs	# Issuers	Owner	Description
1012744	2014-05-19	8	8	Firmaprofesional	Publicly disclosed subordinate CA certificates from Firmaprofesional
1013081	2014-05-20	6	6	ACCV	ACCV publicly disclosed subordinate CA certificates
1016347	2014-05-27	3	3	ACEDICOM	Publicly disclosed subordinate CA certificates from ACEDICOM
1017583	2014-05-29	40	27	GlobalSign	Public disclosure of GlobalSign Subordinate CAs
1018158	2014-05-30	9	9	GRCA	GRCA publicly disclosed subordinate CA certificates
1309707	2016-10-12	7	6	WoSign	Distrust new certs chaining up to current WoSign/StartCom roots
1367842	2017-05-25	3	3	TurkTrust	TurkTrust: Non-audited, non-technically-constrained intermediate certs
1368171	2017-05-26	2	2	Firmaprofesional	Firmaprofesional: Non-audited, non-technically-constrained intermediate certs
1368176	2017-05-26	7	5	DigiCert	DigiCert: Non-audited, non-technically-constrained intermediate certs
1368178	2017-05-26	1	1	Symantec	Symantec: Non-audited, non-technically-constrained intermediate cert
1373452	2017-06-15	3	2	TrustID	Identrust TrustID Subordinate CA - Revocation Notification
1386891	2017-08-02	2	2	StartCom	Certinomis: Cross-signing of StartCom intermediate certs, and delay in reporting it in CCADB
1432608	2018-01-23	15	4	Gov. of Portugal (SCEE)	Add EC Raiz Estado Cross Certificates to OneCRL
1451950	2018-04-05	2	2	Gov. of Portugal (SCEE)	DigiCert: Intermediate Cert(s) not disclosed in CCADB
1451953	2018-04-05	4	4	TeliaSonera	TeliaSonera: Intermediate Cert(s) Not Disclosed in CCADB
1455119	2018-04-18	2	2	Firmaprofesional	Firmaprofesional: Undisclosed Intermediate certificate
1455128	2018-04-18	4	2	Certicamara	Certicamara: Undisclosed Intermediate certificates
1455132	2018-04-18	13	13	SwissSign	SwissSign: Undisclosed Intermediate Certificates
1455137	2018-04-18	1	1	T-Systems	T-Systems: Undisclosed Intermediate certificate
1464359	2018-05-25	1	1	Firmaprofesional	Firmaprofesional: Undisclosed Intermediate certificate SDS
1497700	2018-10-09	1	1	DocuSign/Keynectis	DocuSign/Keynectis: Undisclosed Intermediate certificate
1497703	2018-10-09	2	2	SECOM	SECOM: Undisclosed intermediate certificates
1499585	2018-10-16	26	21	DigiCert	Digicert: Undisclosed CAs -Federated Trust CA-1
1503638	2018-10-31	1	1	WISeKey	WISeKey: Failure to disclose intermediate in CCADB
1542082	2019-04-04	1	1	IdenTrust	Identrust: Failure to disclose Unconstrained intermediate Within 7 Days
1563573	2019-07-04	22	22	DigiCert	DigiCert: Failure to disclose Unconstrained Intermediate within 7 Days
1563574	2019-07-04	2	2	SECOM	SECOM: Failure to disclose Unconstrained Intermediate within 7 Days
1563575	2019-07-04	1	1	TeliaSonera	Telia: Failure to disclose Unconstrained Intermediate within 7 Days
Total: 28	–	186	150	21	–

Table A.1: **CCADB disclosure issues**—28 resolved disclosure issues provide an approximate ground truth dataset of 150 issuers (186 certificates) for `Fides` evaluation.

APPENDIX B: ISSUERS WITH MULTIPLE CCADB OWNERS

Issuer (Subject+SPKI)	CCADB owners	# Certs	Details
ec38da6:MULTICERT SSL CA 005	AC Camerfirma, S.A. | MULTICERT	2	Undisclosed / unaudited MULTI-CERT sub-CA
49d8519:Starfield Services Root CA - G2	Amazon Trust Services | GoDaddy	3	Undisclosed Amazon sub-CA
98ac41c:StartCom Class 3 OV Server CA	StartCom | WoSign	2	Undisclosed StartCom sub-CA
5e87566:Belgium Root CA4	Certipost s.a./n.v. | DigiCert	3	Undisclosed Certipost sub-CA
d42c25d:Let's Encrypt Authority X1	IdenTrust | ISRG	3	Undisclosed ISRG sub-CA
dafa2be:Let's Encrypt Authority X2	IdenTrust | ISRG	3	Undisclosed ISRG sub-CA
78d2913:Let's Encrypt Authority X3	IdenTrust | ISRG	2	Undisclosed ISRG sub-CA
fdeacfa:Let's Encrypt Authority X4	IdenTrust | ISRG	2	Undisclosed ISRG sub-CA
6ee23dd:SSL.com EV Root CA RSA R2	Asseco | SSL.com	3	Disclosed SSL.com sub-CA
39904e6:SSL.com Root CA RSA	Asseco | SSL.com	2	Disclosed SSL.com sub-CA
51b64a7:UCA Global G2 Root	Asseco | Shanghai Elec. CA	2	Disclosed SHECA sub-CA
7712fbc:MULTICERT SSL CA 001	AC Camerfirma, S.A. | MULTICERT	3	Disclosed MULTICERT sub-CA
fa2de6c:GTS Root R1	GlobalSign | Google Trust Services	2	Disclosed GTS sub-CA
2da3659:DigiCert High Assurance EV Root CA	DigiCert | Entrust	6	Expired Entrust cross-sign
4098e01:Network Solutions CA	Sectigo | Web.com	8	Expired Sectigo cross-signs
df6609e:Government CA	Certipost s.a./n.v. | DigiCert	2	Expired / undisclosed DigiCert cross-sign
67d2813:Government CA	Certipost s.a./n.v. | DigiCert	2	Expired / undisclosed DigiCert cross-sign
219718a:Federal Bridge CA 2013	DigiCert | IdenTrust | US Federal PKI	3	All revoked
4c76dcf:Actalis Authentication CA G2	Actalis | DigiCert	3	All revoked
1fb3270:Certipost E-Trust Primary Normalised CA	Certipost s.a./n.v. | DigiCert	2	All revoked
ed0fa26:ECRaizEstado	DigiCert | Gov. of Portugal (SCEE)	6	Revoked DigiCert cross-signs
5b5804f:CA of WoSign G2	Asseco | WoSign	3	Revoked Asseco cross-signs
c23714e:Belgium Root CA2	DigiCert | GlobalSign	3	Revoked GlobalSign cross-sign
5c78ccd:WellsSecure Public Root CA	DigiCert | Wells Fargo Bank N.A.	2	Revoked DigiCert cross-sign
1fc94be:WellsSecure Public Root CA 01 G2	DigiCert | Wells Fargo Bank N.A.	3	Revoked DigiCert cross-signs
5451b03:AffirmTrust Commercial	Entrust | SwissSign AG	2	Revoked SwissSign cross-sign
8b7b0ab:AffirmTrust Networking	Entrust | SwissSign AG	3	Revoked SwissSign cross-signs
69286df:GlobalSign Root CA	GlobalSign | Google Trust Services	2	Revoked GTS cross-sign
1ac0e91:GlobalSign	GlobalSign | Google Trust Services	3	Revoked GlobalSign cross-signs
e125939:CA of WoSign	Sectigo | StartCom | WoSign	6	Revoked StarCom/Sectigo cross-signs
676cf22:CA Wotong Root Certificate	StartCom | WoSign	3	Revoked StartCom cross-signs
b53b021:Microsoft Azure TLS Issuing CA 06	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
4002521:Microsoft Azure ECC TLS Issuing CA 01	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
c0f4b26:Microsoft Azure TLS Issuing CA 05	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
0c6cbcf:Microsoft Azure TLS Issuing CA 02	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
04026ad:Microsoft Azure TLS Issuing CA 01	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
6664c4c:Microsoft Azure ECC TLS Issuing CA 02	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
52929fe:Microsoft Azure ECC TLS Issuing CA 06	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign
e6b1b8a:Microsoft Azure ECC TLS Issuing CA 05	DigiCert | Microsoft Corporation	2	Revoked DigiCert cross-sign

Table B.1: **Issuers with multiple CCADB owners—**39 issuers (Subject+SPKI) across 110 certificates have ambiguous CCADB owners, which reflect CCADB's unsuitability for mapping CA certificate control.

APPENDIX C: 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	Google product based on Chromium OS. Excluded because no build target history.
	Debian	Yes	Mature Linux distribution, 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 [197]	No	Defaults to root store.
	BoringSSL [198]	No	Google fork of OpenSSL used in Chrome/Chromium/Android.
	Bouncy Castle [199]	No	No default, requires configured keystore.
	cryptlib [200]	No	Unknown default.
	GnuTLS [201]	No	Configuration via `--with-default-trust-store-<format>` flag.
	Java Secure Socket Ext. (JSSE) [202]	Yes	`cacerts` JKS file.
	LibreSSL libtls/libssl [203]	No	Configured `TLS_DEFAULT_CA_FILE`.
	MatrixSSL [204]	No	No default, requires configuration.
	Mbed TLS (prev. PolarSSL) [205]	No	No default, requires configuration of `ca_path/ca_file`.
	Network Security Services (NSS) [113]	Yes	Root store in `certdata.txt`, add'l trust elsewhere [116].
	OkHttp [206]	No	Uses platform (e.g., JSSE, BouncyCastle, etc.) TLS.
	OpenSSL [207]	No	Defaults to `$OPENSSLDIR/{certs, cert.pem}`, often symlinked by installer to system certs for Linux. Windows / macOS
	RSA BSAFE [208]	No	Unknown default.
	S2n [209]	No	Defaults to system stores.
	SChannel [151]	No	Defaults to system (Microsoft) store.
	wolfSSL (prev. CyaSSL) [210]	No	No default, requires configuration.
	Erlang/OTP SSL [211]	No	Unknown default.
	BearSSL [212]	No	No default, requires configuration.
	NodeJS [112]	Yes	Static file `src/node_root_certs.h`.
TLS Clients	Safari	No	Uses macOS root store[1].
	Mobile Safari	No	Uses iOS root store.
	Chrome	Yes*	Historically used system roots, with browser control of EV and special cases (e.g., Symantec [81]). As of December 2020, the Chrome root store [125] was deployed on ChromeOS and Linux, with full transition pending [213].
	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 [127]. 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 C.1: **Popular OS & TLS Software Root Stores**

APPENDIX D: ROOT PROGRAM EXCLUSIVE DIFFERENCES

Cert SHA256	CA	NSS inclusion?	Details
NSS (1)			
beb00b30...	Microsec	Accepted [214]	New elliptic curve root.
Java (0)	–	–	–
Apple (13)			
0ed3ffab...	Gov. of Venezuela	Denied [215]	Microsoft trusts issuer for email, disallowed on 2020-02 (PSPProcert). Failed NSS inclusion for same issuer due to super CA concerns.
9f974446...	Certipost	–	CA revoked cross-sign [216]: 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 [217]	Trusted by Microsoft and NSS for email.
cb627d18...	DigiCert	Accepted [217]	Trusted by Microsoft and NSS for email.
a1a86d04...	D-TRUST	Accepted [218]	Microsoft/NSS trusted for email.
5 roots	*Apple*	–	*Roots for custom Apple Services (e.g., FairPlay, Developer ID)*
Microsoft (30)			
1501f89c...	EDICOM	Denied [219]	Inadequate audits, issuance concerns, CA unresponsiveness.
416b1f9e...	e-monitoring.at	Denied [220]	CA certificate violations of the BRs and RFC 5280.
6e0bff06...	Gov. of Brazil	Denied [221]	Super CA concerns, insufficient auditing / disclosure.
c795ff8f...	Gov. of Tunisia	Denied [222]	Repeated misissuance exposed during public discussion.
407c276b...	Gov. of Korea	Denied [223]	Rejected due to confidential, unrestrained subCAs.
c1d80ce4...	AC Camerfirma	Denied [224]	Many issues [225] lead to May 2021 removal of all Camerfirma roots.
ad016f95...	PostSignum	Abandoned [226]	New PostSignum root inclusion attempt running into issues [227].
7a77c6c6...	OATI	Abandoned [228]	No response in 3 years.
604d32d0...	MULTICERT	Abandoned [229]	External subCA concerns and other misissuance. MULTICERT intermediate distrusted in Camerfirma removal [225].
e2809772...	Digidentity	Retracted [230]	
2e44102a...	Gov. of Tunisia	Pending [231]	Community concerns about added-value of the root.
e74fbda5...	SECOM	Pending [232]	Pending since 2016 due to ongoing issue resolution.
24a55c2a...	SECOM	Pending [232]	Pending since 2016 due to ongoing issue resolution.
f015ce3c...	Chunghwa Telecom	Pending [233]	
5ab4fcdb...	Fina	Pending [234]	
242b6974...	Telia	Pending [235]	<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 [236].
229ccc19...	Cisco	–	<100 leaf certificates in CT. NSS rejected older root issuing local certificates shipped with Cisco devices [237].
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 via Baltimore CyberTrust Root.
1a0d2044...	Sectigo	–	Apple/NSS trusted issuer through different root certificate.
3 roots	*Asseco/e-monitoring.at*	*Approved [238, 239]*	*Recently approved by NSS, awaiting addition.*

Table D.1: **Root Store Differences**—Java and NSS rarely implement unique trust, while Apple and Microsoft display more permissive inclusion.

APPENDIX E: NSS ROOT REMOVALS

Bugzilla ID	**Severity**	**Removed On**	**# Certs**	**Details**
1552374	high	2019-07-05	1	Certinomis removal [119]
1392849	high	2017-11-14	3	StarCom removal [135]
1408080	high	2017-11-14	1	PSPProcert removal [132]
1387260	high	2017-11-14	4	WoSign removal [120]
1380868	high	2017-07-27	2	CNNIC removal [134]
682927	high	2011-10-06	1	DigiNotar removal [133]
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 E.1: **NSS Root Removals—**High and medium severity removals from NSS since 2010.

REFERENCES

[1] J. R. Prins, “DigiNotar Certificate Authority breach “Operation Black Tulip”,” 2011.

[2] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, “The Matter of Heartbleed,” in *14th ACM Internet Measurement Conference*, Nov 2014.

[3] Z. Durumeric, E. Wustrow, and J. A. Halderman, “Zmap: Fast internet-wide scanning and its security applications,” in *22nd USENIX Security Symposium*, 2013.

[4] B. Laurie, A. Langley, and E. Kasper, “Certificate Transparency,” RFC 6962, Jun 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6962

[5] J. Clark and P. C. Van Oorschot, “Sok: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements,” in *34th IEEE Symposium on Security and Privacy*, 2013.

[6] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, “Imperfect forward secrecy: How Diffie-Hellman fails in practice,” in *22nd ACM Conference on Computer and Communications Security*, 2015.

[7] B. Möller, T. Duong, and K. Kotowicz, “This POODLE bites: Exploiting the SSL 3.0 fallback,” https://www.openssl.org/~bodo/ssl-poodle.pdf, 2014.

[8] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni et al., “DROWN: Breaking TLS using sslv2,” in *25th USENIX Security Symposium*, 2016.

[9] J. Rizzo and T. Duong, “The CRIME attack,” https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1d134dff_1_222.

[10] Y. Gluck, N. Harris, and A. Prado, “Breach: Reviving the crime attack,” http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf.

[11] J. Rizzo and T. Duong, “Browser exploit against SSL/TLS,” http://packetstormsecurity.com/files/105499/Browser-Exploit-Against-SSL-TLS.html, 2011.

[12] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, “Multiple handshakes security of TLS 1.3 candidates,” in *37th IEEE Symposium on Security and Privacy*, 2016.

[13] K. Bhargavan and G. Leurent, “Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH,” in *23rd Network & Distributed Systems Symposium*, 2016.

[14] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, “Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication,” in *37th IEEE Symposium on Security and Privacy*, 2016.

[15] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub, “Triple handshakes and cookie cutters: Breaking and fixing authentication over tls,” in *35th IEEE Symposium on Security and Privacy*, 2014.

[16] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, “Cryptographically verified implementations for TLS,” in *15th ACM Conference on Computer and Communications Security*, 2008.

[17] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub, “Implementing TLS with verified cryptographic security,” in *34th IEEE Symposium on Security and Privacy*, 2013.

[18] K. Bhargavan, B. Blanchet, and N. Kobeissi, “Verified models and reference implementations for the TLS 1.3 standard candidate,” in *38th IEEE Symposium on Security and Privacy*, 2017.

[19] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, “A comprehensive symbolic analysis of TLS 1.3,” in *24th ACM Conference on Computer and Communications Security*, 2017.

[20] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue, “Implementing and proving the TLS 1.3 record layer,” in *38th IEEE Symposium on Security and Privacy*, 2017.

[21] S. Y. Chau, O. Chowdhury, E. Hoque, H. Ge, A. Kate, C. Nita-Rotaru, and N. Li, “Symcerts: Practical symbolic execution for exposing noncompliance in X.509 certificate validation implementations,” in *38th IEEE Symposium on Security and Privacy*, 2017.

[22] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, “Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations,” in *35th IEEE Symposium on Security and Privacy*, 2014.

[23] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, “A messy state of the union: Taming the composite state machines of TLS,” in *36th IEEE Symposium on Security and Privacy*, 2015.

[24] J. De Ruiter and E. Poll, “Protocol state fuzzing of TLS implementations,” in *24th USENIX Security Symposium*, 2015.

[25] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana, “Hvlearn: Automated blackbox analysis of hostname verification in SSL/TLS implementations,” in *38th IEEE Symposium on Security and Privacy*, 2017.

[26] D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, and L. Khan, “Smv-hunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps,” in *21st Network & Distributed Systems Symposium*, 2014.

[27] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, “To pin or not to pin—helping app developers bullet proof their TLS connections,” in *24th USENIX Security Symposium*, 2015.

[28] M. Oltrogge, N. Huaman, S. Amft, Y. Acar, M. Backes, and S. Fahl, “Why eve and mallory still love android: Revisiting TLS (in) security in android applications,” in *30th USENIX Security Symposium*, 2021.

[29] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, “The most dangerous code in the world: validating SSL certificates in non-browser software,” in *19th ACM Conference on Computer and Communications Security*, 2012.

[30] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar, “Tls in the wild: An internet-wide analysis of TLS-based protocols for electronic communication,” 2015.

[31] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman, “Towards a complete view of the certificate ecosystem,” in *16th ACM Internet Measurement Conference*, 2016.

[32] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, “The SSL landscape: A thorough analysis of the X.509 PKI using active and passive measurements,” in *11th ACM Internet Measurement Conference*, 2011.

[33] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, “Analysis of the HTTPS certificate ecosystem,” in *13th ACM Internet Measurement Conference*, 2013.

[34] J. Amann, R. Sommer, M. Vallentin, and S. Hall, “No attack necessary: The surprising dynamics of ssl trust relationships,” in *29th Annual Computer Security Applications Conference*, 2013.

[35] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, “Mission accomplished? HTTPS security after DigiNotar,” in *17th ACM Internet Measurement Conference*, 2017.

[36] T. Chung, Y. Liu, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, “Measuring and applying invalid ssl certificates: the silent majority,” in *16th ACM Internet Measurement Conference*, 2016.

[37] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, “Bamboozling certificate authorities with BGP,” in *27th USENIX Security Symposium*, 2018.

[38] J. Hiller, J. Amann, and O. Hohlfeld, “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*, 2020.

[39] C. Forum, “Baseline requirements,” https://cabforum.org/baseline-requirements-documents/.

[40] P. Bowen, “Certlint,” https://github.com/awslabs/certlint.

[41] K. Roeckx, “X509lint,” https://github.com/kroeckx/x509lint.

[42] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, “Tracking certificate misissuance in the wild,” in *39th IEEE Symposium on Security and Privacy*, 2018.

[43] A. Delignat-Lavaud, M. Abadi, A. Birrell, I. Mironov, T. Wobber, and Y. Xie, “Web pki: Closing the gap between guidelines and practices.” in *21st Network & Distributed Systems Symposium*, 2014.

[44] E. Stark, R. Sleevi, R. Muminovic, D. O’Brien, E. Messeri, A. P. Felt, B. McMillion, and P. Tabriz, “Does certificate transparency break the web? measuring adoption and error rate,” in *40th IEEE Symposium on Security and Privacy*, 2019.

[45] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch, “The rise of Certificate Transparency and its implications on the internet ecosystem,” in *18th ACM Internet Measurement Conference*, 2018.

[46] R. Roberts, Y. Goldschlag, R. Walter, T. Chung, A. Mislove, and D. Levin, “You are who you appear to be: A longitudinal study of domain impersonation in TLS certificates,” in *26th ACM Conference on Computer and Communications Security*, 2019.

[47] P. Kintis, N. Miramirkhani, C. Lever, Y. Chen, R. Romero-Gómez, N. Pitropakis, N. Nikiforakis, and M. Antonakakis, “Hiding in plain sight: A longitudinal study of combosquatting abuse,” in *24th ACM Conference on Computer and Communications Security*, 2017.

[48] U. Meyer and V. Drury, “Certified phishing: Taking a look at public key certificates of phishing websites,” in *15th Symposium on Usable Privacy and Security (SOUPS)*, 2019.

[49] R. Roberts and D. Levin, “When Certificate Transparency is too transparent: Analyzing information leakage in HTTPS domain names,” in *18th ACM Workshop on Privacy in the Electronic Society (WPES)*, 2019.

[50] P. Hallam-Baker, R. Stradling, and B. Laurie, “DNS Certification Authority Authorization (CAA) Resource Record,” RFC 8659, Nov 2019. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8659

[51] Q. Scheitle, T. Chung, J. Hiller, O. Gasser, J. Naab, R. van Rijswijk-Deij, O. Hohlfeld, R. Holz, D. Choffnes, A. Mislove et al., “A first look at certification authority authorization (caa),” 2018.

[52] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla et al., “Let’s encrypt: An automated certificate authority to encrypt the entire web,” in *26th ACM Conference on Computer and Communications Security*, 2019.

[53] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housely, and P. W., “Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile,” RFC 5280, May 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280

[54] N. Vallina-Rodriguez, J. Amann, C. Kreibich, N. Weaver, and V. Paxson, “A tangled mass: The Android root certificate stores,” in *10th ACM Conference on emerging Networking Experiments and Technologies*, 2014.

[55] N. Korzhitskii and N. Carlsson, “Characterizing the root landscape of Certificate Transparency logs,” in *IFIP Networking Conference (Networking)*, 2020.

[56] L. Waked, M. Mannan, and A. Youssef, “To intercept or not to intercept: Analyzing TLS interception in network appliances,” in *Asia Conference on Computer and Communications Security*, 2018.

[57] J. Braun and G. Rynkowski, “The potential of an individualized set of trusted CAs: Defending against CA failures in the Web PKI,” in *International Conference on Social Computing*. IEEE, 2013.

[58] H. Perl, S. Fahl, and M. Smith, “You won’t be needing these any more: On removing unused certificates from trust stores,” in *International Conference on Financial Cryptography and Data Security*, 2014.

[59] J. Kasten, E. Wustrow, and J. A. Halderman, “CAge: Taming certificate authorities by inferring restricted scopes,” in *International Conference on Financial Cryptography and Data Security*, 2013.

[60] I. Dacosta, M. Ahamad, and P. Traynor, “Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties,” in *European Symposium on Research in Computer Security*, 2012.

[61] S. Matsumoto, J. Bosamiya, Y. Dai, P. van Oorschot, and B. Parno, “Caps: Smoothly transitioning to a more resilient web PKI,” in *Annual Computer Security Applications Conference*, 2020.

[62] Z. Ma, “asn1-fingerprint,” https://github.com/zzma/asn1-fingerprint.

[63] Z. Durumeric, “tlsfingerprints,” https://github.com/zakird/tlsfingerprints.

[64] “Fides source code/data,” https://github.com/zzma/ca-transparency.

[65] U. Inc., “Web archive: GeoTrust and USERTrust offering free SSL service to businesses worldwide,” https://web.archive.org/web/20020802152743/http://www.usertrust.com/index.asp?key=news/free-ssl-service.

[66] R. Miller, “VeriSign to buy GeoTrust, combining top SSL providers,” https://news.netcraft.com/archives/2006/05/17/verisign_to_buy_geotrust_combining_top_ssl_providers.html.

[67] “Notice regarding Symantec acquisition of Verisign’s authentication businesses,” https://www.verisign.com/en_US/verisign-repository/symantec/index.xhtml.

[68] DigiCert, "DigiCert completes acquisition of Symantec's Website Security and related PKI solutions," https://www.digicert.com/news/digicert-completes-acquisition-of-symantec-ssl/.

[69] Comodo, "Secure faxing. secure e-mail. secure backup - anywhere, anytime." https://www.comodo.com/news/press_releases/12_01_04.html.

[70] Comodo, "Comodo CA is now Sectigo," https://comodosslstore.com/sectigo.

[71] R. Sleevi, "Disclosure and CP/CPS for cross-signed roots," https://groups.google.com/d/msg/mozilla.dev.security.policy/89iF_4Ovpwg/zboFW5c6DwAJ.

[72] L. Network Solutions, "Network solutions certification practice statemen," https://assets.web.com/legal/English/CertificationPracticeStatement.pdf.

[73] J. Aas, "Let's encrypt is trusted," https://letsencrypt.org/2015/10/19/lets-encrypt-is-trusted.html.

[74] Mozilla, "Common CA Database," https://www.ccadb.org/.

[75] Symantec, "Symantec acquires PGP and GuardianEdge," http://eval.symantec.com/mktginfo/enterprise/other_resources/b-pgp_guardianedge_acq_faq.en-us.pdf.

[76] "GeoTrust acquires Equifax's digital certificate business," https://www.bizjournals.com/atlanta/stories/2001/09/24/daily17.html.

[77] F. Universe, "VeriSign, Inc. history," http://www.fundinguniverse.com/company-histories/verisign-inc-history/.

[78] F. Universe, "RSA Security Inc. history," http://www.fundinguniverse.com/company-histories/rsa-security-inc-history/.

[79] C. developers, "Symantec roots," https://chromium.googlesource.com/chromium/src/+/master/net/data/ssl/symantec/README.md.

[80] Mozilla Wiki, "CA:Symantec Issues," https://wiki.mozilla.org/CA:Symantec_Issues.

[81] D. O'Brien, R. Sleevi, and A. Whalley, "Chrome plan to distrust Symantec certificates," https://security.googleblog.com/2017/09/chromes-plan-to-distrust-symantec.html.

[82] G. Markham, "Mozilla's plan for Symantec roots," https://groups.google.com/forum/#!msg/mozilla.dev.security.policy/FLHRT79e3XE/riCrpXsfAgAJ.

[83] Apple, "Information for website operators about distrusting Symantec certificate authorities," https://support.apple.com/en-us/HT208860.

[84] M. S. B. Staff, "Microsoft partners with digicert to begin deprecating symantec tls certificates," https://www.microsoft.com/security/blog/2018/10/04/microsoft-partners-with-digicert-to-begin-deprecating-symantec-tls-certificates/.

[85] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An end-to-end measurement of certificate revocation in the web's pki," in *15th ACM Internet Measurement Conference*, 2015.

[86] J. Rowley, https://groups.google.com/d/msg/mozilla.dev.security.policy/_EnH2IeuZtw/AdZvpzGJAwAJ.

[87] DigiCert, "Digicert completes purchase of quovadis, expands european presence and tls, pki offerings," https://www.digicert.com/news/pr/digicert-completes-purchase-of-quovadis-ssl/.

[88] Google, "Chrome Compliant CT logs," https://www.certificate-transparency.org/known-logs.

[89] Apple, "Current Apple CT logs," https://valid.apple.com/ct/log_list/current_log_list.json.

[90] P. Hadfield, "Comment on retrieving, storing and querying 250m+ certificates like a boss," https://medium.com/@hadfieldp/hey-ryan-c0fee84b5c39.

[91] O. Systems, "Asn.1: Listing of universal tags," https://obj-sys.com/asn1tutorial/node124.html.

[92] Let's Encrypt, "Boulder: An ACME-based certificate authority, written in Go." https://github.com/letsencrypt/boulder.

[93] Certipost, "Belgian certificate policy & practice statement for eid pki infrastructure citizen ca," http://repository.eid.belgium.be/downloads/citizen/archive/en/CITIZEN_CA_2018.pdf.

[94] L. Schellman & Company, "Identrust - 2019 webtrust for cas," https://www.cpacanada.ca/generichandlers/CPACHandler.ashx?attachmentid=236834.

[95] Bugzilla, "Camerfirma: Failure to abide by section 8 of Mozilla policy: Unauthorized, improperly disclosed subordinate CA," https://bugzilla.mozilla.org/show_bug.cgi?id=1672029.

[96] B. Wilson, "Summary of Camerfirma's compliance issues," https://groups.google.com/g/mozilla.dev.security.policy/c/dSeD3dgnpzk.

[97] Ernst & Young LLP, "Sectigo: Report of independent accountants," https://bug1472993.bmoattachments.org/attachment.cgi?id=9078178.

[98] Ernst & Young LLP, "Web.com: Report of independent accountants," https://www.cpacanada.ca/generichandlers/CPACHandler.ashx?attachmentid=230861.

[99] PWC, "Certipost independent assurance report," https://bug1461443.bmoattachments.org/attachment.cgi?id=9149485, 2020.

[100] BSI, "Digidentity b.v. audit," https://www.digidentity.eu/assets/files/terms/20200123-ETS043-411-1.pdf, 2020.

[101] R. Hurst, “Google trust services roots,” https://groups.google.com/g/mozilla.dev.security.policy/c/1PDQv0GUW_s/m/ErxcjAcFDwAJ.

[102] J. Kozolchyk, “How to prepare for aws’s move to its own certificate authority,” https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/.

[103] Mozilla Wiki, “CA/Root store policy archive,” https://wiki.mozilla.org/CA/Root_Store_Policy_Archive.

[104] J. Bohm, “General issues that came up in the darkmatter discussion(s),” https://groups.google.com/g/mozilla.dev.security.policy/c/7WuWS_20758/m/erK0-f0GCwAJ.

[105] “Docker hub: alpine,” https://hub.docker.com/_/alpine/.

[106] “Docker hub: amazonlinux,” https://hub.docker.com/_/amazonlinux.

[107] “Android ca-certificates,” https://android.googlesource.com/platform/system/ca-certificates.

[108] “Secure Transport,” https://opensource.apple.com/source/Security/.

[109] “Debian ca-certificates,” https://salsa.debian.org/debian/ca-certificates.

[110] “OpenJDK source,” https://github.com/openjdk/.

[111] R. Stradling, “authroot.stl,” https://github.com/robstradling/authroot.stl.

[112] “NodeJS,” https://github.com/nodejs/node.

[113] “Network Security Services (NSS) Mercurial repository,” https://hg.mozilla.org/projects/nss.

[114] “Ubuntu ca-certificates,” https://launchpad.net/ubuntu/+source/ca-certificates.

[115] “Mozilla ca/faq,” https://wiki.mozilla.org/CA/FAQ.

[116] “CA/Additional Trust Changes,” https://wiki.mozilla.org/CA/Additional_Trust_Changes.

[117] “Google groups: mozilla.dev.security.policy,” https://groups.google.com/g/mozilla.dev.security.policy.

[118] “Google groups: dev-security-policy@mozilla.org,” https://groups.google.com/a/mozilla.org/g/dev-security-policy.

[119] “Ca/certinomis issues,” https://wiki.mozilla.org/CA/Certinomis_Issues.

[120] “Ca:wosign issues,” https://wiki.mozilla.org/CA:WoSign_Issues.

[121] W. Thayer, “DarkMatter Concerns,” https://groups.google.com/g/mozilla.dev.security.policy/c/nnLVNfqgz7g/m/TseYqDzaDAAJ.

[122] Mozilla, "WoSign and StartCom," https://docs.google.com/document/d/1C6BlmbeQfn4a9zydVi2UvjBGv6szuSB4sMYUcVrR8vQ/edit.

[123] "Windows root certificate program members," https://web.archive.org/web/20110728002957/http://support.microsoft.com/kb/931125, 2010.

[124] "Apple root certificate program," https://web.archive.org/web/20050503225244/http://www.apple.com/certificateauthority/ca_program.html, May 2005.

[125] "Chrome root program," https://www.chromium.org/Home/chromium-security/root-ca-policy.

[126] "Java SE CA root certificate program," https://www.oracle.com/java/technologies/javase/carootcertsprogram.html.

[127] "Root certificates used by Opera," https://web.archive.org/web/20150207210358/http://www.opera.com/docs/ca/, 2015.

[128] "Electron's chromium is trusting different CAs then Electron's NodeJS," https://github.com/electron/electron/issues/11741, 2018.

[129] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The Security Impact of HTTPS Interception," in *24th Network & Distributed Systems Symposium*, 2017.

[130] 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, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

[131] "Super-CAs," https://wiki.mozilla.org/CA/Subordinate_CA_Checklist\#Super-CAs.

[132] "CA:PROCERT issues," https://wiki.mozilla.org/CA:PROCERT_Issues.

[133] J. Nightingale, "Diginotar removal follow up," https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/, September 2011.

[134] "The MCS incident and its consequences for CNNIC," https://blog.mozilla.org/security/files/2015/04/CNNIC-MCS.pdf, April 2015.

[135] K. Wilson, "Distrusting new WoSign and StartCom certificates," https://blog.mozilla.org/security/2016/10/24/distrusting-new-wosign-and-startcom-certificates/, 2016.

[136] "CA/Certinomis issues," https://wiki.mozilla.org/CA/Certinomis_Issues.

[137] "Removed CA Certificate List," https://ccadb-public.secure.force.com/mozilla/RemovedCACertificateReport.

[138] H. Adkins, "An update on attempted man-in-the-middle attacks," https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html.

[139] J. Nightingale, "Fraudulent *.google.com certificate," https://blog.mozilla.org/security/2011/08/29/fraudulent-google-com-certificate/, August 2011.

[140] "Security update 2011-005," https://support.apple.com/kb/dl1447, September 2011.

[141] "CNNIC Action Items," https://bugzilla.mozilla.org/show_bug.cgi?id=1177209.

[142] "About the security partial trust allow list," https://support.apple.com/en-gb/HT204938.

[143] "Implement the Symantec distrust plan from Bug 1409257," https://hg.mozilla.org/mozreview/gecko/rev/f6c9341fde050d7079a8934636644aaf54bde922, 2018.

[144] "Symantec root certs - set cka_nss_server_distrust_after," https://bugzilla.mozilla.org/show_bug.cgi?id=1618404.

[145] "ca-certificates should remove Symantec certs," https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=911289.

[146] "ca-certificates: Removal of GeoTrust Global CA requires investigation," https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=962596.

[147] J. Douglas, " Incident: NuGet Restore Issues on Debian Family Linux Distros," https://github.com/NuGet/Announcements/issues/49.

[148] "CAcert root cert inclusion into browser," https://bugzilla.mozilla.org/show_bug.cgi?id=215243.

[149] "Review Request: ca-cacert.org - CAcert.org CA root certificates," https://bugzilla.redhat.com/show_bug.cgi?id=474549.

[150] "crypto: add deprecated ValiCert CA for cross cert," https://github.com/nodejs/node/pull/1135.

[151] Microsoft, "Schannel," https://docs.microsoft.com/en-us/windows/win32/com/schannel.

[152] T. Dierks and E. Rescorla, "Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Tech. Rep., Aug 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5246

[153] C. Evans, C. Palmer, and R. Sleevi, "Public key pinning extension for HTTP," RFC 7469, Tech. Rep., Apr 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7469

[154] A. Langley, "Public key pinning," https://www.imperialviolet.org/2011/05/04/pinning.html.

[155] Microsoft, "Platform status," https://developer.microsoft.com/en-us/microsoft-edge/platform/status/publickeypinningextensionforhttp.

[156] IANA, "Transport layer security (TLS) extensions," http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml.

[157] IANA, “Transport layer security (TLS) parameters,” http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml.

[158] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, “Host fingerprinting and tracking on the web: Privacy and security implications.” in *19th Network & Distributed Systems Symposium*, 2012.

[159] P. Eckersley, “How unique is your web browser?” in *Symposium on Privacy Enhancing Technologies*, 2010.

[160] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, “Cookieless monster: Exploring the ecosystem of web-based device fingerprinting,” in *34th IEEE Symposium on Security and Privacy*, 2013.

[161] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, “Fingerprinting information in JavaScript implementations,” *Web 2.0 Security & Privacy*, 2011.

[162] U.S. Digital Analytics Program, “The U.S. federal government’s web traffic,” https://analytics.usa.gov/.

[163] R. Arora and N. Aggarwal, “Browserstack,” https://browserstack.com.

[164] R. Seggelmann, M. Tuexen, and M. Williams, “Transport layer security (TLS) and datagram transport layer security (DTLS) heartbeat extension,” RFC 6520, Tech. Rep., Feb 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6520

[165] Mozilla, “Network security services (NSS),” https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS.

[166] E. Bursztein, “Speeding up and strengthening HTTPS connections for Chrome on Android,” https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html.

[167] H. Böck, “Superfishy,” https://github.com/hannob/superfishy.

[168] Komodia, “Redirector,” http://www.komodia.com/products/komodia-redirector.

[169] W. Dormann, “The risks of SSL inspection,” https://insights.sei.cmu.edu/cert/2015/03/the-risks-of-ssl-inspection.html.

[170] X. de Carné de Carnavalet and M. Mannan, “Killed by proxy: Analyzing client-end TLS interception software,” in *23rd Network & Distributed Systems Symposium*, 2016.

[171] M. O’Neill, S. Ruoti, K. Seamons, and D. Zappala, “TLS proxies: Friend or foe?” in *16th ACM Internet Measurement Conference*, 2016.

[172] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, “Analyzing forged SSL certificates in the wild,” in *35th IEEE Symposium on Security and Privacy*, 2014.

[173] OPSWAT, “Antivirus and compromised device report: Janurary 2015,” https://www.opswat.com/resources/reports/antivirus-and-compromised-device-january-2015.

[174] Symantec, "Trojan.Nurjax," https://www.symantec.com/security_response/writeup.jsp?docid=2014-121000-1027-99.

[175] "NetFilter SDK," http://netfiltersdk.com/.

[176] V. Paxson, "Bro: a system for detecting network intruders in real-time," in *7th USENIX Security Symposium*, 1998.

[177] N. Weaver, C. Kreibich, M. Dam, and V. Paxson, "Here be web proxies," in *International Conference on Passive and Active Network Measurement*, 2014.

[178] J. Graham-Cumming, "The two reasons to be an engineer at Cloudflare," https://blog.cloudflare.com/the-two-reasons-to-work-for-cloudflare/.

[179] Mozilla, "Installing certificates into Firefox," https://wiki.mozilla.org/CA:AddRootToFirefox.

[180] Statista, "Mobile phone internet user penetration in South Korea from 2014 to 2019," 2016, http://www.statista.com/statistics/284204/south-korea-mobile-phone-internet-user-penetration/.

[181] A. Kirk, "Web proxies, user-agent strings, and malware detection," http://blog.talosintel.com/2012/11/web-proxies-user-agent-strings-and.html.

[182] Chromium, "IsSecureTLSCipherSuite function," https://chromium.googlesource.com/chromium/src/net/+/master/ssl/ssl_cipher_suite_names.cc#373.

[183] F. Valsorda, "Komodia/superfish SSL validation is broken," February 2015, https://blog.filippo.io/komodia-superfish-ssl-validation-is-broken/.

[184] Dell, "Information on the eDellRoot and DSDTestProvider certificates," http://www.dell.com/support/article/us/en/19/SLN300321.

[185] T. Ormandy, "Avast antivirus: X.509 error rendering command execution," https://bugs.chromium.org/p/project-zero/issues/detail?id=546&can=1&q=avast.

[186] A. Popov, "Prohibiting RC4 cipher suites," RFC 7465, Tech. Rep., Feb 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7465

[187] M. Vanhoef and F. Piessens, "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS," in *24th USENIX Security Symposium*, 2015.

[188] P. Lepeska, "Comments on explicit/trusted proxy," https://www.ietf.org/mail-archive/web/httpbisa/current/msg13124.html.

[189] D. McGrew, "Comments on explicit/trusted proxy," https://www.ietf.org/mail-archive/web/tls/current/msg07815.html.

[190] Mozilla, "Revoking intermediate certificates: Introducing OneCRL," https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/.

[191] Google Chromium, “CRLSets,” https://dev.chromium.org/Home/chromium-security/crlsets.

[192] “OpenSSL changelog,” https://www.openssl.org/news/changelog.html.

[193] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman, “Neither snow nor rain nor MITM...: An empirical analysis of email delivery security,” in *15th ACM Internet Measurement Conference*, 2015.

[194] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, “Tracking the FREAK attack,” https://freakattack.com/.

[195] M. Ray, S. Dispensa, and E. Rescorla, “Transport Layer Security (TLS) Renegotiation Indication Extension,” RFC 5746, Tech. Rep., Feb 2010. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5746

[196] Y. N. Pettersen, “The Transport Layer Security (TLS) Multiple Certificate Status Request Extension,” RFC 6961, Jun 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6961

[197] “Botan: Crypto and TLS for Modern C++,” https://github.com/randombit/botan.

[198] “BoringSSL,” https://boringssl.googlesource.com/boringssl/.

[199] “Bouncy Castle,” http://git.bouncycastle.org/index.html.

[200] “cryptlib,” https://www.cs.auckland.ac.nz/~pgut001/cryptlib/.

[201] “GnuTLS,” https://gitlab.com/gnutls/gnutls/blob/master/README.md.

[202] “OpenJDK,” http://hg.openjdk.java.net/.

[203] “LibreSSL libtls,” https://cvsweb.openbsd.org/src/lib/libtls/.

[204] “MatrixSSL,” https://github.com/matrixssl/matrixssl.

[205] “Mbed TLS,” https://github.com/ARMmbed/mbedtls.

[206] “OkHttp,” https://github.com/square/okhttp.

[207] “OpenSSL,” https://github.com/openssl/openssl.

[208] “RSA BSAFE,” https://community.rsa.com/community/products/bsafe.

[209] “s2n,” https://github.com/awslabs/s2n.

[210] “wolfSSL,” https://github.com/wolfSSL/wolfssl.

[211] “Erlang OTP SSL,” https://github.com/erlang/otp/tree/master/lib/ssl.

[212] "BearSSL," https://bearssl.org/.

[213] R. Sleevi, "Announcing the chrome root program," https://groups.google.com/g/mozilla.dev.security.policy/c/3Q36J4flnQs/m/VyWFiVwrBQAJ.

[214] "Microsec new (ecc) root inclusion request," https://bugzilla.mozilla.org/show_bug.cgi?id=1445364.

[215] "Add Autoridad de Certificacion Raiz del Estado Venezolano root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1302431.

[216] "Add DigiCert non-TLS Intermediate Certs to OneCRL," https://bugzilla.mozilla.org/show_bug.cgi?id=1404501.

[217] "Add Symantec-brand Class 1 and Class 2 roots," https://bugzilla.mozilla.org/show_bug.cgi?id=833986.

[218] "Add D-TRUST Root CA 3 2013 to NSS," https://bugzilla.mozilla.org/show_bug.cgi?id=1348132.

[219] "Add Renewed ACEDICOM root certificate(s)," https://bugzilla.mozilla.org/show_bug.cgi?id=1239329.

[220] "Add GLOBALTRUST 2015 root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1440271.

[221] "Add CA Root certificate (Brazil's National PKI)," https://bugzilla.mozilla.org/show_bug.cgi?id=438825.

[222] "Add TunRootCA2 root certificate(s)," https://bugzilla.mozilla.org/show_bug.cgi?id=1233645.

[223] "Add MOI GPKI Root CA certificate(s)," https://bugzilla.mozilla.org/show_bug.cgi?id=1226100.

[224] "Add Renewed AC Camerfirma root certificate." https://bugzilla.mozilla.org/show_bug.cgi?id=986854.

[225] "Ca:camerfirma issues," https://wiki.mozilla.org/CA:Camerfirma_Issues.

[226] "Add PostSignum root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=643398.

[227] "Add PostSignum Root QCA 4 to Root Store," https://bugzilla.mozilla.org/show_bug.cgi?id=1602415.

[228] "Add OATI's Root CA Certificate to Mozilla's trusted root list," https://bugzilla.mozilla.org/show_bug.cgi?id=848766.

[229] "Add MULTICERT Root Certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1040072.

[230] "Add Digidentity Service Root Certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1558450.

[231] "Add TunTrust Root CA root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1587779.

[232] "Add 2 new SECOM root certificates," https://bugzilla.mozilla.org/show_bug.cgi?id=1313982.

[233] "Add Chunghwa Telecom's HiPKI Root CA -G1 Certificate to NSS," https://bugzilla.mozilla.org/show_bug.cgi?id=1563417.

[234] "Add "Fina Root CA" root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1449941.

[235] "Add Telia CA root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1664161.

[236] "add Finnish Population Register Centre's Root CA Certificates," https://bugzilla.mozilla.org/show_bug.cgi?id=463989.

[237] "Add Cisco Root CA Cert," https://bugzilla.mozilla.org/show_bug.cgi?id=416842.

[238] "Add e-commerce monitoring's GLOBALTRUST 2020 root certificate," https://bugzilla.mozilla.org/show_bug.cgi?id=1627552.

[239] "Add Asseco DS / Certum root certificates," https://bugzilla.mozilla.org/show_bug.cgi?id=1598577.