Server Authentication on the Past, Present, and Future Internet

by

James Douglas Kasten, Jr.

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2015

Doctoral Committee:

Associate Professor J. Alex Halderman, Chair
Professor Mingyan Liu
Assistant Professor Harsha V. Madhyastha
Professor Atul Prakash

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Server Authentication on the Past, Present and Future Internet

by

James Douglas Kasten, Jr.

Chair: J. Alex Halderman

HTTPS is used for nearly all secure web communication, yet very little is known about the security of HTTPS' deployment overall on the Internet. In this work, we elucidate the efficacy of HTTPS' security through Internet-wide scanning and present novel solutions for some of the most critical issues we discover.

Our analysis includes the first longitudinal study of the HTTPS ecosystem, and a study of the HTTPS ecosystem during upheaval, including the community's subsequent response. This examination revealed not only the common practices, but also a number of alarming trends. In this thesis, we focus on two of these issues. The first is that the PKI underlying HTTPS has an extremely large attack surface, with 683 organizations able to sign certificates for any domain. The second is that the cost of HTTPS is exorbitant. As evidence, we found that only 12.9% of the Alexa Top 1 Million supported HTTPS and that 55% of servers with browser-trusted certificates are not optimally configured. Furthermore, we find the management of HTTPS is too burdensome. We discover 20% of certificates are removed from servers after they have already expired.

In order to address the large attack surface of the PKI, we present CAge. CAge is a technique that can reduce the attack surface of certificate authorities by 90% using simple

inference techniques. The key observation is that CAs commonly sign for only a handful of TLDs; in fact, 90% of CAs have signed certificates for domains in fewer than 10 TLDs, and only 35% have ever signed a certificate for a domain in `.com`.

To decrease the cost of HTTPS, we present Let's Encrypt, the first fully automated and free certificate authority. The automation is enabled by a new protocol we developed, ACME, which handles all of a CA's operational duties. We implement client and server ACME software which reduces the time required to deploy HTTPS to 30 seconds. We additionally develop new validation techniques which improve the security of the PKI in general.

Thesis Statement: Measurement-based security and automation can reduce the vulnerabilities originating from both certificate authority practice and HTTPS server deployments.

CHAPTER I

Introduction

TLS has become the narrow waist of the Internet for secure online transactions. Every day, millions of Internet users rely on HTTPS to connect securely to online services such as banking, e-mail, and e-commerce. However, most normal web traffic remains in cleartext over HTTP [114].

This normal HTTP traffic is subject to two classes of network attacks: eavesdropping and man-in-the-middle (MITM) attacks. Eavesdropping acknowledges the fact that the content is not encrypted and that any party along the path of the communication can read its content. Eavesdropping has obvious privacy implications, and it enables attacks like session hijacking [108]. MITM attackers can both intercept and modify content. MITM attackers can insert trojans in software downloads [54], modify webpages to perform DDoS attacks [57], inject advertisements [77], and add cookies to users' requests to track them [95].

Given all of the problems with HTTP and the recent revelations of global surveillance, there has been a major push to deprecate HTTP altogether in favor of HTTPS [26]. Users around the world have learned to associate the browser's HTTPS lock icon with security, but few users understand the implications and guarantees provided [130].

HTTPS helps to protect against passive and active attacks. HTTPS combines the Transport Layer Security (TLS) protocol with a public-key infrastructure (PKI) based on certificate authorities (CAs) that are trusted by the browser. When clients connect to a server

over TLS, the server presents its public key in the form of an X.509 certificate. The certificate ties the domain name to a public key and is digitally signed by a CA. The CA is responsible for verifying the identity of the website, usually for a small fee. Browsers maintain a set of trusted root CAs and subsequently trust the purported identities of certificates signed by any CA in this trusted set. In addition, these root CAs are typically able to sign certificates for additional CAs, known as intermediate certificate authorities, which are trusted recursively by the browser.

CA-signed certificates provide authentication and protect users in the presence of man-in-the-middle adversaries. If certificate authorities are secure, trustworthy, and properly verify the identity of websites before issuing certificates, it should be impossible to attain and present a CA-signed certificate for a domain that you do not control. If an attacker attempts to submit an invalid or untrusted certificate, the user is issued a strong warning urging them to leave the site. In recent years the warnings have become increasingly effective at deterring users from continuing onto the compromised site, requiring successful attackers to subvert the CA system itself [19]. HTTPS' authentication and security directly relies on CAs performing their role correctly.

Unfortunately, it is impossible to determine if the CAs are performing in their expected capacity. The difficulty of appraising the certificate ecosystem arises from the offline nature of certificate signing and the chaining of intermediate certificates. The client can only become aware of certificates when they are presented. Users of the system are unable to monitor the certificate authorities' activities or even determine the full set of certificate authorities that they trust.

In order to understand this normally opaque security-critical infrastructure, we scanned the Internet and retrieved all of the certificates available on port 443, the standard port used for HTTPS. In Chapter II, "Analysis of the HTTPS Certificate Ecosystem", we present the first full systematic analysis of HTTPS certificates, including results from 110 scans over 14 months. This analysis granted us a new perspective on the health and state of PKI on the

Internet.

To glean an additional viewpoint on the ecosystem, we studied HTTPS system administrators' ability to maintain the security of their systems. In April of 2014, the Heartbleed vulnerability was discovered within OpenSSL which left 24–55% of the HTTPS servers on the Internet vulnerable. We scanned for the vulnerability and monitored the community's response, including the steps taken to rectify their systems. The Heartbleed analysis gave us a unique perspective on operational security of HTTPS deployments. These two analyses have uncovered the following pervasive problems with HTTPS, which we will address in this thesis.

Massive Attack Surface

We found that 683 different organizations could sign trusted certificates for any domain on the Internet. This presents an extremely large attack surface as attackers only have to compromise the weakest CA to break the security of TLS. In recent years, there have been several high-profile attacks [78, 112] and CA blunders [81] that resulted in the signing of fraudulent certificates. For instance, in 2011, an attacker breached the security of a relatively small Dutch CA named DigiNotar and created certificates for dozens of popular sites, including *.google.com [8]. An ISP in Iran subsequently abused this latter certificate to conduct man-in-the-middle attacks against Google services [29]. We attempt to constrain compromised CAs in Chapter IV on CAge.

Misconfigured Servers

We discovered sites that deploy HTTPS are often misconfigured. 12.7% of servers presented invalid certificate chains from trusted CAs, and only 45% were optimally configured. This causes problems for the stability, reliability, and security of the servers for end-users. Servers suffering from expired or misconfigured certificates will often create meaningless errors for clients or potentially cover up more troublesome problems. Regardless of the errors, they cause users to lose trust in the domain and the certificate warning system in general.

Slow Patching Rate

Even if servers have initially been setup correctly, there are often necessary updates to the software and configuration in order to maintain proper security. CRIME [98], BEAST [115], Lucky 13 [20], Heartbleed [96], ShellShock [6], POODLE [40], FREAK [5], and Logjam [17] all required attention and updates by affected parties. In our study of Heartbleed [42] and POODLE [40], we found that system administrators are slow to update their systems, and frequently do so incorrectly. Less than a quarter of the Alexa Top 1 Million sites that were vulnerable to Heartbleed replaced their certificates within the first week. Additionally, 14% did not change their private key, thus providing no security benefit.

Low Adoption

HTTPS adoption, as a whole, remains low, even for popular websites. We found only 12.9% of the Alexa Top 1 Million sites supported HTTPS in our scans. HTTP has outlived its usefulness. Governments have been exploiting HTTP by collecting web traffic information *en masse* and exploiting packet-injection vulnerabilities [54, 86]. Packet injection has been used by China to insert malicious JavaScript into traffic to perform a distributed-denial-of-service (DDoS) attack against sites performing censorship resistance [57]. Packet injection has been used to exploit browser vulnerabilities, which grant the attacker full access to the victim's computer [116]. Packet injection has also been used to impersonate the victim's target website [54]. Governments are not the only guilty parties that have exploited HTTP, corporations have also been exploiting the weaknesses afforded by HTTP. Verizon was discovered inserting persistent tracking cookies into their customers' HTTP headers [95]. Verizon claimed no harm was done, but inserting tracking headers greatly eases mass surveillance and increases the risk of being tracked by all websites the user visits. Advertising companies were found utilizing the Verizon tracker to target their markets [94]. Comcast has been known to inject ads into their customer's visited websites [77] and companies

have colluded with Internet service providers for user search queries [55]. All of these security and privacy concerns are mitigated by HTTPS. HTTPS has been found to be an effective tool to hinder surveillance, stop censorship, and increase attribution. The largest problem with HTTPS is that it is not deployed widely enough.

Based on the findings from our measurement research, we go on to propose a series of mitigations and new systems that aim to improve the security of the Internet as a whole. In Chapter IV we tackled the problem of the CAs' *massive attack surface*. By further investigating CAs' signing behavior, we determined that many individual CAs only sign certificates for domains within a small number of TLDs. We developed metrics and algorithms to automatically constrain CAs based on their past signing behavior. The technique reduces the attack surface of the HTTPS PKI by over 90% (by one metric). In fact, had the system been put in place before the DigiNotar hack [29], it would have prevented 300,000 users from having their Gmail accounts attacked.

Low adoption makes it clear that the cost of adoption is too high. In purely economic terms, the cost of adopting TLS outweighs the benefits for many system administrators. Surveys and investigation into the process yielded the two largest costs of deploying TLS: the time required to deploy TLS and the monetary cost of the certificate. The difficulty of deploying TLS correlates well with our evidence of the *slow patching rate* and *misconfigured servers* in our prior analysis.

The way we deploy HTTPS is fundamentally broken and does not allow for the widescale adoption the Internet needs. The Internet needs a "free" certificate authority; free both monetarily and in terms of system administrator time. Servers need to default to HTTPS, and not require any additional human interaction to serve HTTPS. In Chapter V, we present Let's Encrypt, the first completely free and automated certificate authority that is trusted by all major browsers. We developed a new protocol, Automated Certificate Management Environment (ACME), which allows the certificate requestor's interaction with the CA to be fully automatic. IETF has formed a working-group to make ACME a formal standard

and ACME will be used by Let's Encrypt. We have open-sourced both client and server implementations of ACME and expect further adoption by other CAs. In addition to solving the original problems we set out to resolve with ACME, we also found solutions for other long-standing PKI problems.

1.1 Summary of Main Contributions

Analysis of the HTTPS Certificate Ecosystem

We report the results of the first large-scale measurement study of the HTTPS certificate ecosystem. Using data collected by performing 110 Internet-wide scans over 14 months, we gain detailed and temporally fine-grained visibility into this otherwise opaque area of security-critical infrastructure. We investigate the trust relationships among root authorities, intermediate authorities, and the leaf certificates used by web servers, ultimately identifying and classifying more than 1,800 entities that are able to issue certificates vouching for the identity of any website. We uncover practices that may put the security of the ecosystem at risk, and we identify frequent configuration problems that lead to user-facing errors and potential vulnerabilities.

Analysis of HTTPS Updates and Patches

The Heartbleed vulnerability took the Internet by surprise in April 2014. The vulnerability, one of the most consequential since the advent of the commercial Internet, allowed attackers to remotely read protected memory from an estimated 24-55% of popular HTTPS sites. We monitored the community as they dealt with Heartbleed and patched their systems. This illuminated problems with the deployment of HTTPS, namely, the ability of system administrators to patch their systems effectively in a timely manner. System administrators showed a clear willingness to patch their systems, but oftentimes did so incorrectly.

Minimizing Certificate Authorities Attack Surface

The existing HTTPS public-key infrastructure (PKI) uses a coarse-grained trust model: either a certificate authority (CA) is trusted by browsers to vouch for the identity of any domain or it is not trusted at all. More than a thousand root and intermediate CAs can currently sign certificates for any domain and be trusted by popular browsers. This violates the principle of least privilege and creates an excessively large attack surface, as highlighted by recent CA compromises. We present CAge, a mechanism that browser makers can apply to drastically reduce the excessive trust placed in CAs without fundamentally altering the CA ecosystem or breaking existing practice. CAge works by imposing restrictions on the set of top-level domains (TLDs) for which each CA is trusted to sign. Our key observation, based on an Internet-wide survey of TLS certificates, is that CAs commonly sign for only a handful of TLDs; in fact, 90% of CAs have signed certificates for domains in fewer than 10 TLDs, and only 35% have ever signed a certificate for a domain in .com. We show that it is possible to algorithmically infer reasonable restrictions on CAs' trusted scopes based on this behavior, and we present evidence that browser-enforced inferred scopes would be a durable and effective way to reduce the attack surface of the HTTPS PKI. We find that simple inference rules can reduce the attack surface by nearly a factor of ten without hindering 99% of CA signing activity over a six-month period.

Let's Encrypt: A certificate authority to encrypt the entire Internet

Although HTTP has seen tremendous adoption, it is insecure by design. HTTPS offers a base level of confidentiality, authenticity, and integrity, but it has yet to see wide deployment across the Internet at large. At the heart of the problem are current CA practices and deployment issues. We analyze current CA practices, CA marketing, and CA costs, gaining valuable insight into the market at large. In response, we propose and develop Let's Encrypt, the first completely automated and free certificate authority. We develop a new protocol, the Automated Certificate Management Environment (ACME), in order to completely automate all processes of current domain validation

certificate authorities. We examine the bureaucracy, costs, and deployment issues related to becoming a browser-trusted CA. We introduce the first implementations of ACME and the reasoning behind their design. Finally, we conclude by analyzing the industry and community reception. Let's Encrypt is scheduled for general availability to the public the week of November 16, 2015.

This research has illuminated and identified the major problems within TLS. My thesis solves many of the problems within HTTPS's PKI and is currently being adopted by the industry. This work will help the community further understand the inherent practical problems with PKI in untrusted environments, provide valuable insight into solving these problems, and provide a usable mechanism to automatically establish identification and trust. Measurement-based security and automation can reduce the vulnerabilities originating from both certificate authority practice and HTTPS server deployments.

1.2 Structure of Thesis

This thesis is organized into six parts. In Chapter II we illuminate the state of X.509 certificates through analyzing Internet-wide scans. In Chapter III we examine how system administrators respond to massive vulnerabilities within TLS, acquiring new understanding into certificate and TLS operational issues. In Chapter IV we offer a solution to CA signing behavior and the proliferation of certificate authority trust. Finally, in Chapter V, we describe the development and details of Let's Encrypt, an automated certificate authority, to encrypt the entire Internet. We conclude with Chapter VI on future work.

CHAPTER II

Analysis of the HTTPS Certificate Ecosystem [43]

2.1 Introduction

Nearly all secure web communication takes place over HTTPS including online banking, e-mail, and e-commerce transactions. HTTPS is based on the TLS encrypted transport protocol and a supporting public key infrastructure (PKI) composed of thousands of certificate authorities (CAs)—entities that are trusted by users' browsers to vouch for the identity of web servers. CAs do this by signing digital certificates that associate a site's public key with its domain name. We place our full trust in each of these CAs—in general, every CA has the ability to sign trusted certificates for *any* domain, and so the entire PKI is only as secure as the weakest CA. Nevertheless, this complex distributed infrastructure is strikingly opaque. There is no published list of signed website certificates or even of the organizations that have trusted signing ability. In this work, we attempt to rectify this and shed light on the HTTPS certificate ecosystem.

Our study is founded on what is, to the best of our knowledge, the most comprehensive dataset of the HTTPS ecosystem to date. Between June 2012 and August 2013, we completed 110 exhaustive scans of the public IPv4 address space in which we performed TLS handshakes with all hosts publicly serving HTTPS on port 443. Over the course of 14 months, we completed upwards of 400 billion SYN probes and 2.55 billion TLS handshakes, collecting and parsing 42.4 million unique X.509 certificates from 109 million hosts. On

average, each of our scans included 178% more TLS hosts and 115% more certificates than were collected in earlier studies of the certificate authority ecosystem [47], and we collected 736% more unique certificates in total than any prior study of HTTPS [63].

Using this dataset, we investigate two classes of important security questions, which relate to the behavior of CAs and to site certificates.

Certificate Authorities We analyze the organizations involved in the HTTPS ecosystem and identify 1,832 CA certificates, which are controlled by 683 organizations including religious institutions, museums, libraries, and more than 130 corporations and financial institutions. We find that more than 80% of the organizations with a signing certificate are not commercial certificate authorities and further investigate the paths through which organizations are acquiring signing certificates. We investigate the constraints on these CA certificates and find that only 7 CA certificates use name constraints, and more than 40% of CA certificates have no path length constraint. We identify two sets of misissued CA certificates and discuss their impact on the security of the ecosystem.

Site Certificates We analyze leaf certificates used by websites and find that the distribution among authorities is heavily skewed towards a handful of large authorities, with three organizations controlling 75% of all trusted certificates. Disturbingly, we find that the compromise of the private key used by one particular intermediate certificate would require 26% of HTTPS websites to immediately obtain new certificates. We provide an up-to-date analysis on the keys and signatures being used to sign leaf certificates and find that half of trusted leaf certificates contain an inadequately secure 1024-bit RSA key in their trust chain and that CAs were continuing to sign certificates using MD5 as late as April 2013. We find that 5% of trusted certificates are for locally scoped names or private IP address space (and therefore do not protect against man-in-the-middle attacks) and that 12.7% of hosts serving certificates signed by trusted CAs are serving them in a manner that will cause errors in one or more modern web browsers.

Lastly, we examine adoption trends in the HTTPS ecosystem from the past year, discuss anomalies we noticed during our analysis, and provide high-level lessons and potential paths forward to improve the security of the HTTPS ecosystem security. We ultimately hope that this global perspective and our analysis will inform future decisions within the security community as we work towards a more secure PKI. In order to facilitate future research on this critical ecosystem, we are releasing our dataset to the research community, including 42 million certificates and historical records of the state of 109 million HTTPS server IP addresses. This data and up-to-date metrics can be found at https://scans.io/.

2.2 Background

In this section, we present a brief review of TLS, digital certificates and their respective roles within the HTTPS ecosystem. We recommend RFC 5280 [36] for a more in-depth overview of the TLS public key infrastructure.

Transport Layer Security (TLS) Transport Layer Security (TLS) and its predecessor Secure Sockets Layer (SSL) are cryptographic protocols that operate below the application layer and provide end-to-end cryptographic security for a large number of popular application protocols, including HTTPS, IMAPS, SMTP, and XMPP [39]. In the case of HTTPS, when a client first connects, the client and server complete a TLS handshake during which the server presents an X.509 digital certificate, which is used to help identify and authenticate the server to the client. This certificate includes the identity of the server (e.g. website domain), a temporal validity period, a public key, and a digital signature provided by a trusted third party. The client checks that the certificate's identity matches the requested domain name, that the certificate is within its validity period, and that the digital signature of the certificate is valid. The certificate's public key is then used by the client to share a session secret with the server in order to establish an end-to-end cryptographic channel.

Certificate Authorities Certificate authorities (CAs) are trusted organizations that issue digital certificates. These organizations are responsible for validating the identity of the websites for which they provide a digital certificate. They cryptographically vouch for the identity of a website by digitally signing the website's *leaf certificate* using a browser-trusted *signing certificate*. Modern operating systems and web browsers ship with a set of these trusted *signing certificates*, which we refer to as *root certificates*. In all but a small handful of cases, all CAs are trusted unequivocally: a trusted CA can sign for *any* website. For example, a certificate for `google.com` signed by a German University is technically no more or less valid than a certificate signed by Google Inc., if both organizations control a trusted signing certificate.

The set of root authorities is publicly known because it is included with the web browser or operating system. However, root authorities frequently sign *intermediate certificates*, which generally retain all of the signing privileges of root certificates. This practice not only allows root authorities to store their signing keys offline during daily operation, but also allows authorities to delegate their signing ability to other organizations. When a server presents a leaf certificate, it must include the chain of authorities linking the leaf certificate to a trusted root certificate. This bundle of certificates is referred to as a *certificate chain*. We refer to certificates that have a valid chain back to a trusted root authority as *trusted certificates*. It is important to note that while intermediate authorities provide additional flexibility, the set of intermediate authorities is not publicly known until they are found in the wild—we ultimately do not know the identity of the organizations that can sign any browser-trusted certificate.

2.3 Related Work

Several groups have previously studied HTTPS deployment and the certificate ecosystem. Most similar to our work, Holz et al. published a study in 2011 that focused on the dynamics of leaf certificates and the distribution of certificates among IP addresses, and attempted to

roughly classify the overall quality of served certificates. The study was based on regular scans of the Alexa Top 1 Million Domains [1] and through passive monitoring of TLS traffic on the Munich Scientific Research Network [64]. The group collected an average 212,000 certificates per scan and a total 554,292 unique certificates between October 2009 and March 2011, approximately 1.3% of the number we have seen in the past year. Their passive experiments resulted in an average of 130,000 unique certificates. The aggregate size across both datasets was not specified.

We are aware of two groups that have performed scans of the IPv4 address space in order to analyze aspects of the certificate ecosystem. In 2010, the Electronic Frontier Foundation (EFF) and iSEC partners performed a scan over a three-month period as part of their SSL Observatory Project [47]. The project focused on identifying which organizations controlled a valid signing certificate. The EFF provided the first recent glimpse into the HTTPS certificate ecosystem, and while their study was never formally published, we owe the inspiration for our work to their fascinating dataset. Heninger et al. later performed a scan of the IPv4 address space in 2012 as part of a global study on cryptographic keys [63]. Similarly, Yilek et al. performed daily scans of 50,000 TLS servers over several months to track the Debian weak key bug [132]. We follow up on the results provided in these earlier works, adding another data point in the study of Debian weak keys and other poorly generated keys.

Most recently, Akhawe et al. published a study focusing on the usability of TLS warnings presented by web servers, deriving the logic used by web browsers to validate certificates, and making recommendations on how to better handle these error conditions [18]. Akhawe et al. also discuss differences in how OpenSSL and Mozilla NSS validate certificates, which we arrived at simultaneously.

Our study differs from previous work in the methodology we applied, the scope of our dataset, and the focus of our questions. While Holz et al. explored several similar questions on the dynamics of leaf certificates, the dataset we consider is more than 40 times larger,

which we believe provides a more comprehensive view of the certificate ecosystem. The certificates found by scanning the Alexa Top 1 Million Domains provide one perspective on the CA ecosystem that is weighted towards frequently accessed websites. However, many of the questions we address are dependent on a more comprehensive viewpoint. The CA ecosystem is equally dependent on all certificate authorities and, as such, we are interested in not only the most popular sites (which are likely to be well configured) but also the potentially less visible certificates used by smaller sites and network devices. This difference is clearly visible in the number of CA certificates seen among the Alexa Top 1 Million sites. If our study had been founded only on these domains, we would have seen less than 30% of the trusted certificate authorities we uncovered, providing us with a less accurate perspective on the state of the ecosystem. Similarly, we build on many topics touched on by the EFF study, but we present updated and revised results, finding more than 3.5 times the number of hosts serving HTTPS than were seen three years ago and a changed ecosystem. Ultimately, we consider a different set of questions that are more focused on the dynamics of CAs and the certificates they sign, using a dataset that we believe provides a more complete picture than any previous study.

2.4 Methodology

Scan Date Completed	EFF [47] 2010-8	Ps & Qs [63] 2011-10	First 2012-6-10	Representative 2013-3-22	Latest 2013-8-4	Total Unique
Hosts with port 443 Open	16,200,000	28,923,800	31,847,635	33,078,971	36,033,088	(unknown)
Hosts serving HTTPS	7,704,837	12,828,613	18,978,040	21,427,059	24,442,824	108,801,503
Unique Certificates	4,021,766	5,758,254	7,770,385	8,387,200	9,031,798	42,382,241
Unique Trusted Certificates	1,455,391	1,956,267	2,948,397	3,230,359	3,341,637	6,931,223
Alexa Top 1 Mil. Certificates	(unknown)	89,953	116,061	141,231	143,149	261,250
Extd. Validation Certificates	33,916	71,066	89,190	103,170	104,167	186,159

Table 2.1: **Internet-wide Scan Results** — Between June 6, 2012 and August 4, 2013, we completed 110 scans of the IPv4 address space on port 443 and collected HTTPS certificates from responsive hosts.

Our data collection (which is ongoing as this paper goes to press) involves repeatedly

surveying the certificate ecosystem through comprehensive scans of the IPv4 address space conducted at regular intervals. In this section, we describe how we perform these scans, collect and validate X.509 certificates, and finally, analyze our data.

Each scan consists of three stages: (1) discovering hosts with port 443 (HTTPS) open by enumerating the public address space, (2) completing a TLS handshake with responsive addresses and collecting the presented certificate chains, and (3) performing certificate parsing and validation. The scan process requires 18 hours to complete, including flushing all changes to the backend database, and is implemented in approximately 13,000 SLOC of C. The scans in this work were conducted using the regular office network at the University of Michigan Computer Science and Engineering division, from a single Dell Precision workstation with a quad-core Intel Xeon E5520 processor and 24 GB of memory. The access layer of the building runs at 10 Gbps and the building uplink to the rest of the campus is an aggregated 2×10 gigabit port channel.

2.4.1 Host Discovery

In the first stage of each scan, we find hosts that accept TCP connections on port 443 (HTTPS) by performing a single-packet TCP SYN scan of the public IPv4 address space using ZMap [44]. We choose to utilize ZMap based on its performance characteristics—ZMap is capable of completing a single packet scan of the IPv4 address space on a single port in approximately 45 minutes. Using ZMap, we send a single TCP SYN packet to every public IPv4 address and add hosts that respond with a valid SYN-ACK packet to an in-memory Redis queue for further processing. Our previous work finds an approximate 2% packet drop rate when performing single packet scans on our network [44]. In order to reduce the impact of packet loss on our long-term HTTPS results, we also consider hosts that successfully completed a TLS handshake in the last 30 days for follow-up along with the hosts found during the TCP SYN scan.

2.4.2 Collecting TLS Certificates

In the second processing stage, we complete a TLS handshake with the hosts we identified in the first stage and retrieve the presented certificate chain. We perform these TLS handshakes in an event-driven manner using libevent and OpenSSL [93, 127]. Specifically, we utilize libevent's OpenSSL-based bufferevents, which allow us to define a callback that is invoked after a successful OpenSSL TLS negotiation. The retrieval process runs in parallel to the TCP SYN scan and maintains 2,500 concurrent TLS connections.

In order to emulate browser validation, we designed a custom validation process using the root browser stores from Apple Mac OS 10.8.2, Windows 7, and Mozilla Firefox. We find that a large number of web servers are misconfigured and present incomplete, misordered, or invalid certificate chains. OpenSSL validates certificates in a more stringent manner than most web browsers, including Mozilla Firefox and Google Chrome, which utilize Mozilla NSS [101] to perform certificate validation. To simulate the behavior of modern web browsers, we take the following corrective steps:

1. If the presented chain is invalid, we attempt to reorder the certificate chain. This resolves the situation when the correct intermediate certificates are provided, but are in the incorrect order.

2. We add previously seen intermediate authorities into OpenSSL's root store. This allows us to validate any certificate signed by a previously encountered intermediate CA regardless of the presented certificate chain.

3. Following each scan, we check certificates without a known issuer against the set of known authorities and revalidate any children for which there is a newly found issuer. This resolves the case where an intermediate is later found in a subsequent scan.

We parse collected TLS certificates using OpenSSL and maintain a PostgreSQL database of parsed data and historical host state.

2.4.3 Reducing Scan Impact

We recognize that our scans can inadvertently trigger intrusion detection systems and may upset some organizations. Many network administrators perceive port scans as the preliminary step in a targeted attack and in most cases are unable to recognize that their systems are not being uniquely targeted or that our research scans are not malicious in nature.

In order to minimize the impact of our scans and to avoid triggering intrusion detection systems, we scanned addresses according to a random permutation over a twelve hour period from a block of 64 sequential source IP addresses. When we perform a host discovery scan, an individual destination address receives at most one probe packet. At this scan rate, a /24-sized network receives a probe packet every 195 s, a /16 block every 0.76 s, and a /8 network block every 3 ms on average. In the certificate retrieval phase, we perform only one TLS handshake with each host that responded positively during host discovery.

In order to help users identify our intentions, we serve a simple webpage on all of the IP addresses we use for scanning that explains the purpose of our scanning and how to request that hosts be excluded from future scans. We also registered reverse DNS records that identify scanning hosts as being part of an academic research study. Throughout this study, we have coordinated with our local network administrators to promptly handle inquiries and complaints.

Over the course of 14 months, we received e-mail correspondence from 145 individuals and organizations. In most cases, notifications were informative in nature—primarily notifying us that we may have had infected machines—or were civil requests to be excluded from future scans. The vast majority of these requests were received at our institution's WHOIS abuse address or at the e-mail address published on the scanner IPs. In these cases, we responded with the purpose of our scans and excluded the sender's network from future scans upon request. Ultimately, we excluded networks belonging to 91 organizations or individuals and totaling 3,753,899 addresses (0.11% of the public IPv4 address space). Two

requests originating from Internet service providers accounted for 49% of the excluded addresses. During our scans, we received 12 actively hostile responses that threatened to retaliate against our institution legally or via denial-of-service (DoS) attacks on network. In 2 cases we received retaliatory DoS traffic, which was automatically filtered by our upstream provider.

We discuss the ethical implications of performing active scanning and provide more details about the steps we take to reduce scan impact in our previous work [44].

2.4.4 Data Collection Results

We completed 110 successful scans of the IPv4 address space, completing 2.55 billion TLS handshakes, between June 6, 2012 and August 4, 2013. Like to Holz et al. [64], we note that a large number of hosts on port 443 do not complete a TLS handshake. In our case we find that only 67% of hosts with port 443 open successfully complete a TLS handshake.

We retrieved an average of 8.1 million unique certificates during each scan, of which 3.2 million were browser trusted. The remaining 4.9 million untrusted certificates were a combination of self-signed certificates (48%), certificates signed by an unknown issuer (33%), and certificates signed by a known but untrusted issuer (19%). In total, we retrieved 42.4 million distinct certificates from 108.8 million unique IP addresses over the past eleven months. Of the hosts that performed complete TLS handshakes, an average of 48% presented browser-trusted X.509 certificates.

In our largest and most recent scan on August 4, 2013, we retrieved 9.0 million certificates from 24.4 million IP addresses of which 3.3 million were browser trusted. We show a comparison with previous work in Table 2.1. We also note that over 95% of trusted certificates and over 98% of hosts serving trusted certificates are located in only ten countries, shown in Table 2.2.

In this study, we choose to perform non-temporal analysis on the results from a representative scan, which took place on March 22, 2013 (highlighted column in Table 2.1).

Country	Authorities	Certificates	Hosts
United States	30.34%	77.55%	75.63%
United Kingdom	3.27%	10.88%	18.15%
Belgium	2.67%	3.29%	1.51%
Israel	1.63%	2.56%	0.87%
Netherlands	2.18%	1.32%	0.49%
Japan	3.38%	1.06%	1.19%
Germany	21.28%	0.88%	0.35%
France	3.98%	0.38%	0.14%
Australia	0.81%	0.34%	0.11%
Korea	1.41%	0.24%	0.09%

Table 2.2: **Top 10 Countries Serving Trusted Certificates**

We choose to focus on the results from a single point-in-time instead of considering all certificates found over the past year due to varying lifespans. We find that organizations utilize certificates of differing validity periods and that in some cases, some devices have presented a different certificate in all of our scans. If we considered all certificates from the past year instead instead of what was hosted at a single point in time, these short lived certificates would impact the breakdown of several of our statistics.

2.4.5 Is Frequent Scanning Necessary?

Frequent repeated scans allow us to find additional certificates that would not otherwise be visible. We can illustrate this effect by considering the 36 scans we performed between January 1 and March 31, 2013 and analyzing the number of scans in which each certificate was seen. We find that 54% of browser-trusted certificates appeared in all 36 scans and that 70% of trusted certificates appear in more than 30 of our 36 scans. However, surprisingly, we find that 33% of self-signed certificates appeared in only one scan during the three month period. Many of these self-signed certificates appear to be served by embedded devices that generate new certificates on a regular basis. We found an average of 260,000 new certificates per scan during this period. The distribution is shown in Figure 2.1. Ultimately, we find that there are considerable advantages to scanning more frequently in obtaining a global

perspective on the certificates valid at any single point in time, as well as the changing dynamics of the ecosystem over extended periods.

2.4.6 Server Name Indication Deployment

Both Holz [64] and Akhawe [18] cite Server Name Indication (SNI) as one of the reasons they choose to scan the Alexa Top 1 Million Domains and perform passive measurement instead of performing full IPv4 scans. Server Name Indication is a TLS extension that allows a client to specify the hostname it is attempting to connect to from the start of the TLS negotiation [30]. This allows a server to present multiple certificates on a single IP address and to ultimately host multiple HTTPS sites off of the same IP address that do not share a single certificate. Because we connect to hosts based on IP address in our scans and not by hostname, we would potentially miss any certificates that require a specific hostname.

In order to better understand the deployment of SNI and its impact on our results, we scanned the Alexa 1 Million Domains [1] using the same methodology we used for scanning the IPv4 address space. Of the Alexa Top 1 Million Domains, 323,502 successfully performed TLS handshakes and 129,695 of the domains presented browser-trusted certificates. Of the domains that completed a TLS handshake, only 0.7% presented certificates we had not previously seen in the most recent scan of the IPv4 address space. We cannot bound the number of hosts missed due to the deployment of SNI and it is clear that a small number of websites are adopting SNI, but we believe that our results are representative of certificate usage patterns. One reason SNI has not seen widespread deployment is because Internet Explorer on Windows XP does not support SNI. Although Windows XP market share is on the decline, it still represents more than a third of all operating system installations [100].

2.5 Certificate Authorities

The security of the HTTPS ecosystem is ultimately dependent on the set of CAs that are entrusted to sign browser-trusted certificates. Except in a small handful of cases, any

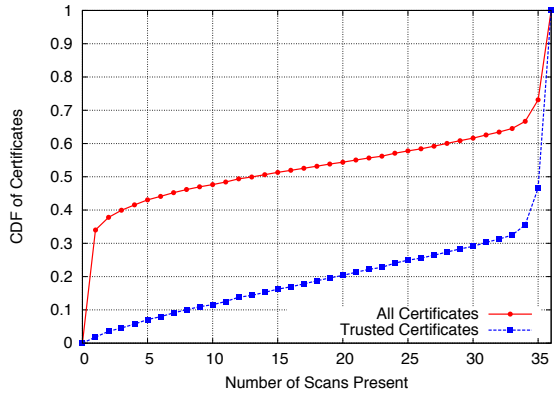

Figure 2.1: **CDF of Scan Presence by Certificate** — We performed 36 scans from 1/2013 to 3/2013. Here, we show the number of scans in which each certificate was found. We note that over 30% of self-signed certificates were only found in one scan.

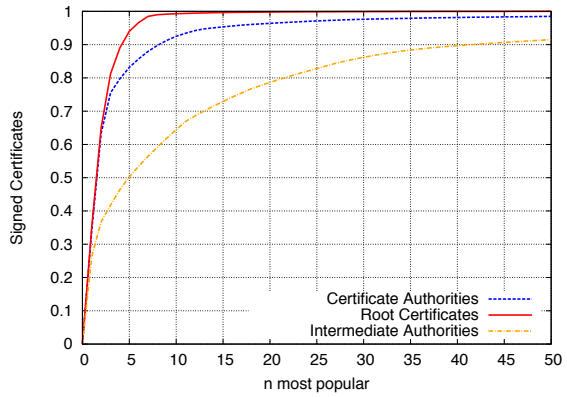

Figure 2.2: **CDF of Leaf Certificates by CA** — We find that 90% of trusted certificates are signed by 5 CAs, are descendants of 4 root certificates, and were signed by 40 intermediate certificates.

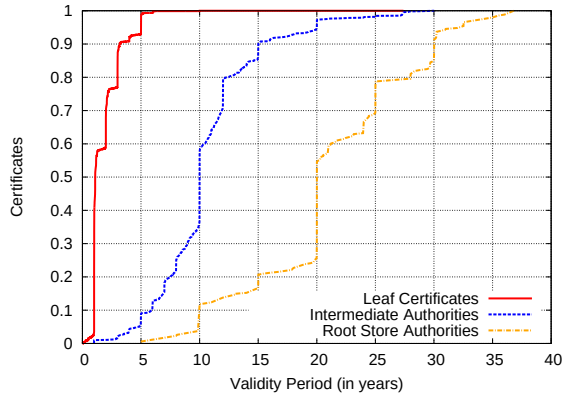

Figure 2.3: **Validity Periods of Browser Trusted Certificates** — Trusted CA certs are being issued with validity periods as long as 40 years, far beyond the predicted security of the keys they contain.

organization with control of a signing certificate that chains to a browser-trusted root can sign a leaf certificate for any domain. As such, the entire ecosystem is as fragile as the weakest CA. However, because there is no central, public registry of browser-trusted intermediate authorities, the organizations that control these signing certificates may be unknown until certificates they have signed are spotted in the wild. In this section, we describe the CAs we found during our scans and some of the practices they employ.

2.5.1 Identifying Trusted Authorities

We observed 3,788 browser-trusted signing certificates between April 2012 and August 2013 of which 1,832 were valid on March 22, 2013. All but seven of these signing certificates can sign a valid browser-trusted certificate for any domain. This is 25% more than were found by the EFF in 2010 and more than 327% more than were found by Ristic [113]. Holz et al. find 2,300 intermediate certificates in their active scanning [64]. However, this count appears to represent both browser-trusted and untrusted intermediates, of which we find 121,580 in our March 22 scan and 417,970 over the past year. While the raw number of signing certificates and HTTPS ecosystem as a whole have grown significantly over the past three years, we were encouraged to find that the number of identified organizations has not grown significantly.

These 1,832 signing certificates belong to 683 organizations and are located in 57 countries. While a large number of countries have jurisdiction over at least one trusted browser authority, 99% of the authorities are located in only 10 countries. We show the breakdown in Table 2.2. We classified the types of the organizations that control a CA certificate, which we show in Table 2.3. We were surprised to find that religious institutions, museums, libraries, and more than 130 corporations and financial institutions currently control an unrestricted CA certificate. Only 20% of organizations that control signing certificates are commercial CAs. We were unable to identify 15 signing certificates due to a lack of identification information or ambiguous naming. We also note that while there

has been a 2% increase in the raw number of valid signing certificates over the past year, we have found negligible change in the number of organizations with control of a signing certificate.

2.5.2 Sources of Intermediates

Organizations other than commercial CAs control 1,350 of the 1,832 (74%) browser-trusted signing certificates, which raises the question of who is providing intermediate certificates to these organizations. We find that 276 of the 293 academic institutions along with all of the libraries, museums, healthcare providers, and religious institutions were signed by the German National Research and Education Network (DFN), which offers intermediate certificates to all members of the German network. DFN provided CA certificates to 311 organizations in total, close to half of the organizations we identified. While DFN has provided a large number of intermediate authorities to German institutions, we find no evidence that any are being used inappropriately. However, as we will discuss in Section 2.9, the attack surface of the certificate ecosystem could be greatly reduced by limiting the scope of these signing certificates.

The largest commercial provider of intermediate certificates is GTE CyberTrust Solutions, Inc., a subsidiary of Verizon Business, which has provided intermediate signing certificates to 49 third-party organizations ranging from Dell Inc. to Louisiana State University. Comodo (under the name *The USERTRUST Network*) provided intermediates to 42 organizations and GlobalSign to 20. We also saw a number of commercial authorities that provided a smaller number of certificates to seemingly unrelated entities. For example, VeriSign, Inc. provided intermediates for Oracle, Symantec, and the U.S. Government; SwissSign AG provided certificates for Nestle, Trend Micro, and other Swiss companies; StartCom Ltd. provided certificates for The City of Osmio, Inc. and WoSign, Inc; QuoVadis Limited provided certificates for Migros and the Arab Bank Switzerland Ltd.; Entrust.net provided signing certificates to Disney, Experian PLC, and TDC Internet; and Equifax

provided intermediates to Google Inc. This is not a clandestine practice, and several CAs advertise the sale of subordinate CA certificates.

Several corporations had a company authority in browser root stores. Approximately 30 of the 149 certificates in the Mozilla NSS root store belonged to institutions that we did not classify as commercial CAs, including Visa, Wells Fargo, Deutsche Telekom AG and the governments of France, Taiwan, Hong Kong, Japan, Spain, and the United States.

2.5.3 Distribution of Trust

While there are 683 organizations with the ability to sign browser-trusted certificates, the distribution is heavily skewed towards a small number of large commercial authorities in the United States. The security community has previously expressed concern over the sheer number of signing certificates [47], but it also worth considering the distribution of certificates among various authorities. An increasing number of signing certificates may in fact be a healthy sign if it indicates that authorities are using the new certificates in order to reduce the impact of compromise.

As shown in Figure 2.2, we find that more than 90% of browser-trusted certificates are signed by the 10 largest commercial CAs, are descendants of just 4 root certificates, and are directly signed by 40 intermediate certificates. Several large companies have acquired many of the smaller, previously independent commercial CAs. Symantec owns Equifax, GeoTrust, TC TrustCenter, Thawte, and VeriSign; GoDaddy owns Starfield Technologies and ValiCert; and Comodo owns AddTrust AB, eBiz Networks, Positive Software, RegisterFly, Registry Pro, The Code Project, The USERTRUST Network, WebSpace-Forum e.K., and Wotone Communications. These consolidations ultimately allow three organizations (Symantec, GoDaddy, and Comodo) to control 75% of the browser-trusted certificates seen in our study. We list the top 10 parent organizations in Table 2.4 and the top 10 commercial CAs in Table 2.5.

There is a long history of commercial CA compromise [29, 112, 120]. In each of these

cases, web browsers and operating systems explicitly blacklisted the compromised signing certificate or misissued certificates [29, 102]. However, if a compromised signing certificate had signed for a substantial portion of the Internet, it would potentially be infeasible to revoke it without causing significant disruption to the HTTPS ecosystem [88]. As such, we would hope that large commercial authorities would distribute signing among a number of intermediate certificates. However, as seen in Figure 2.2, the exact opposite is true. More than 50% of all browser-trusted certificates have been directly signed by 5 intermediate certificates and a single intermediate certificate has signed 26% of currently valid HTTPS certs. If the private key for this intermediate authority were compromised, 26% of websites that rely on HTTPS would need to be immediately issued new certificates. Until these websites deployed the new certificates, browsers would present certificate warnings for all HTTPS communication. While it is not technically worrisome that a small number of organizations control a large percentage of the CA market, it is worrying that large CAs are not following simple precautions and are instead signing a large number of leaf certificates using a small number of intermediates.

2.5.4 Browser Root Certificate Stores

Microsoft, Apple, and Mozilla all maintain a distinct set of trusted signing certificates, which we refer to as root authorities. Google Chrome utilizes the OS root store in Windows and Mac OS and utilizes the root store maintained by Mozilla on Linux. Combined, the three groups trust 348 root authorities, but there are large discrepancies between the root certificates trusted by each organization. For example, as can be seen in Table 2.6, Windows trusts 125 additional authorities that are not present in any other OS or browser.

The differences in the root stores lead to 463 partially trusted CAs. All but a small handful of the partially trusted authorities belong to government, regional, or specialty issuers. Only one of the partially trusted CAs, *ipsCA*, advertised itself as a commercial authority and sold certificates to the global market. Incidentally, the company claims to

be "recognized by more than 98% of today's desktops" [10]. It fails to mention that its certificates are not trusted in Mozilla Firefox or on Mac OS.

Further investigation indicates that *ipsCA* was in the Mozilla root store in 2009, but was removed after several violations including the issuance of embedded-null prefix certificates, the unavailability of OCSP servers, and the issuance of leaf certificates with validity periods beyond the lifetime of the root CA certificate [131].

These 463 partially trusted authorities have little presence on the Internet. In total, they have signed certificates for only 51 domains in the Alexa Top 1 Million and for one domain in the Alexa Top 10,000 which belongs to *mci.ir*, an Iranian telecommunications company. Of the 348 root certificates, 121 of the authorities never signed any leaf certificates seen in our study, and 99.4% of the leaf certificates trusted by any browser are trusted in all browsers.

2.5.5 Name Constraints

While it is not an inherently poor idea to provide signing certificates to third-party organizations, these certificates should be restricted to a limited set of domains. Instead, all but 7 CAs in our March 22 scan can sign for any domain. X.509 Name Constraints [65] provide a technical mechanism by which parent authorities can limit the domains for which an intermediate signing certificate can sign leaf certificates. Optimally, signing certificates provided to third-party organizations, such as universities or corporations, would utilize name constraints to prevent potential abuse and to limit the potential damage if the signing certificate were compromised.

We find that only 7 trusted intermediate authorities out of 1,832 have name constraints defined, of which 3 were labeled as Comodo testing certificates. The remaining 4 are:

1. An intermediate provided by AddTrust AB to the Intel is limited to small a number of Intel owned domains.
2. An intermediate controlled by the U.S. State Department and provided by the U.S.

Government root authority is prevented from signing certificates with the `.mil` top-level domain.

3. An intermediate provided to the Louisiana State University Health System is limited to a small number of affiliated domains.
4. A root certificate belonging to the Hellenic Academic and Research Institutions Certification Authority is restricted to the .gr, .eu, .edu, and .org domains.

2.5.6 Path Length Constraints

A signing authority can limit the number of intermediate authorities that can appear below it in a certificate chain by specifying an X.509 path length constraint [65] on the intermediate certificates that it signs. This is frequently used to prevent intermediate authorities from further delegating the ability to sign new certificates.

In our dataset, we find that 43% of signing certificates do not have any path length restriction defined. While this may not be a concern for large commercial CAs, we note that more than 80% of the intermediate authorities belonging to other types of organizations (e.g. corporations, academic, and financial institutions). While we saw little evidence of non-commercial CAs providing signing certificates to third-party organizations, we did observe governments using their intermediate authority to sign subordinate CA certificates for corporations within their country.

2.5.7 Authority Key Usage

All of the browser-trusted leaf certificates in our study were signed using an RSA key. As shown in Table 2.8, over 95% of browser trusted certificates were signed with 2048-bit RSA keys. We also note 6 browser-trusted authorities with ECDSA keys belonging to Symantec, Comodo, and Trend Micro. However, we found no trusted certificates that were signed using a ECDSA certificate.

Surprisingly, we find that 243 (13%) of the browser-trusted signing certificates were

signed using a weaker key than they themselves contained. In all of these cases, the weakest key was the root authority. While only 58 (15.2%) of the 348 browser root authorities utilize 1024-bit RSA keys, these keys were used to indirectly sign 48.7% of browser-trusted certificates. In all of these cases, the CA organization also controlled a browser-trusted 2048-bit root certificate that could be used to re-sign the intermediate certificate.

NIST recommends that the public stop using 1024-bit keys in 2016 based on the expected computational power needed to compromise keys of this strength [24]. However, as seen in Figure 2.5, more than 70% of CA certificates using 1024-bit keys expire after this date and 57% of roots using 1024-bit RSA keys have signed children that expire after 2016. Figure 2.3 shows how certificate authorities are using certificates valid for up to 40 years—far beyond when their keys are expected to be compromisable. Most worryingly, it does not appear that CAs are moving from 1024-bit roots to more secure keys. As shown in Figure 2.4, we find only a 0.08% decrease in the number of certificates dependent on a 1024-bit root authority in the past year. In 2012, 1.4 million new certificates were issued that were rooted in a 1024-bit authority, and 370,130 were issued between January and April 2013.

2.6 Leaf Certificates and Hosting

Over the last 14 months, we collected 6.93 million unique trusted certificates. In our March 22 scan, we observed 3.2 million unique trusted certificates from 21.4 million hosts. In this section, we discuss the dynamics of these trusted leaf certificates and the hosts serving them.

2.6.1 Keys and Signatures

Public Keys In line with previous studies, we find that over 99% of trusted leaf certificates contain RSA public keys. We provide a breakdown of leaf key types in Table 2.9. Over the course of the past year, we found 47 certificates that contain ECDSA public keys; none were present in our March 22 scan and none were browser trusted. Recently, Google began to use

ECDSA certificates for several services. However, these sites are only accessible through the use of server name indication (SNI) and so do not appear in our dataset.

We find 2,631 browser-trusted certificates using 512-bit RSA keys, which are known to be easily factorable, and 73 certificates utilizing 768-bit keys, which have been shown to be factorable with large distributed computing efforts [75]. While a large number of these certificates were found being actively hosted, only 16 have not yet expired or been revoked. No browser-trusted authorities have signed any 512-bit RSA keys since August 27, 2012. We were further encouraged to find that less than 4% of valid trusted certificates used 1024-bit keys.

Weak Keys Previous studies have exposed the use of weak keys in the HTTPS space [63, 84,132]. We revisit several of these measurements and provide up-to-date metrics. Following up on the study performed by Heninger et al. [63], we find that 55,451 certificates contained factorable RSA keys and are served on 63,293 hosts, a 40% decrease in the total percentage of hosts with factorable keys, but only a slight decrease (1.25%) in the raw number of hosts found using factorable keys since 2011. Three of the factorable certificates are browser trusted; the last was signed on August 9, 2012. 2,743 certificates contained a Debian weak key [28], of which 96 were browser trusted, a 34% decrease from 2011 [63]. The last browser-trusted certificate containing a Debian weak key was signed on January 25, 2012.

Signature Algorithms In line with the results presented by Holz et al. [64], we find that 98.7% of browser-trusted certificates are signed using SHA-1 and RSA encryption. We find 22 trusted certificates with MD2-based signatures and 31,325 with MD5 signatures. Due to known weaknesses in these hash functions, no organizations should currently be using them to sign certificates. The last certificate signed with MD5 was issued on April 17, 2013 by Finmeccanica S.p.A., an Italian defense contractor, more than 4 years after Sotirov et al. published "MD5 considered harmful today" [120]. We provide a breakdown of leaf certificate signature types in Table 2.10.

Certificate Depth Similarly to the EFF and Holz et al., we find that the vast majority (98%) of leaf certificates are signed by intermediate authorities one intermediate away from a root authority. However we find that 61 root authorities directly signed 41,000 leaf certificates and that there exist leaf certificates as many as 5 intermediates away from a root authority. All but a handful of the authorities 4 or more intermediates away from a browser-trusted root belonged to agencies within the U.S. Federal Government.

We are not aware of any vulnerabilities created by having a long certificate chain. However, it is worrisome to see leaf certificates directly signed by root authorities, because this indicates that the root signing key is being actively used and may be stored in a network-attached system, raising the risk of compromise. If the signing key were to be compromised, the root certificate could not be replaced without updating all deployed browser installations. If an intermediate authority were used instead to sign these leaf certificates, then it could be replaced by the root authority without requiring browser updates, and the root could be kept offline during day-to-day operation.

2.6.2 Incorrectly Hosted Trusted Certificates

We find that 1.32 million hosts (12.7%) serving once-valid browser-trusted leaf certificates are misconfigured in a manner such that they are inaccessible to some clients or are being hosted beyond their validity period. We show a breakdown of reasons that certificates are invalid in Table 2.11. We note that Mozilla Network Security Services (NSS) [101], the certificate validation library utilized by many browsers, caches previously seen intermediates. Because of this, many certificates with invalid trust chains will appear valid in users' browsers if the intermediate authorities have previously been encountered.

Approximately 5.8% of hosts are serving now-expired certificates, which will be considered invalid by all browsers. We find that 22% of certificates are removed retroactively after their expiration and that 19.5% of revoked certificates are removed after they appear in a certificate revocation list (CRL). We show the distribution of when certificates are

removed from servers in Figure 2.6. Another 42.2% of hosts are providing unnecessary certificates in the presented trust chain. Although this practice has no security implications, these additional certificates provide no benefit to the client and ultimately result in a slight performance degradation.

Holz et al. report that 18% of all certificates are expired. However, this statistic reflected all certificates, over 25% of which are self-signed and would already raise a browser error. We instead consider only certificates signed by browser-trusted authorities, which would otherwise be considered valid.

2.6.3 Invalid Authority Types

We find that 47 (2.6%) of the 1,832 browser-trusted signing certificates are not denoted for signing TLS certificates for use on the web. Of these 47 signing certificates, 28 (60%) are designated for signing Microsoft or Netscape Server Gated Crypto certificates, a now obsolete cryptographic standard that was used in the 1990s in response to U.S. regulation on the export of strong cryptographic standards [111].

The remaining 19 signing certificates are designated for combinations of *Code Signing*, *E-mail Protection*, *TLS Web Client Authentication*, *Time Stamping*, and *Microsoft Encrypted File System*. These intermediate certificates were not found in any browser or operating system root stores but were found being served on public web servers. It does not appear that any of these authorities were signing certificates inappropriately; nobody was attempting to sign a TLS Web Server Authentication certificate using an authority marked for another use. Instead, we found that individuals and organizations were mistakenly using valid code signing and e-mail certificates as the TLS leaf certificate on their websites.

2.6.4 Certificate Revocation

Certificate authorities can denote that previously issued certificates should no longer be trusted by publishing their revocation in a public *certificate revocation list* (CRL). The

location of authority CRLs are listed in each signed certificate. In order to understand why certificates are being revoked, we fetched and parsed the CRLs listed in all browser-trusted certificates. We find that 2.5% of browser-trusted certificates are eventually revoked by their authority. We present a breakdown of revocation reasons in Table 2.12. While RFC 5280 [36] strongly encourages issuers to provide "meaningful" reason codes for CRL entries, we find that 71.7% of issuers who revoked certificates do not provide reasons for any of their revocations.

While 2.5% of certificates are eventually revoked, we find that only 0.3% of hosts presenting certificates in our scan were revoked. We expect that this is because the site operators will request a certificate be revoked and simultaneously remove the certificate from the web server. As can be seen in Figure 2.6, more than 80% of certificates are removed proactively and were not seen again after the time of their revocation.

WebTrust for Certificate Authorities [11], an audit mandated by the three major root stores, requires that authorities maintain an online repository that allows clients to check for certificate revocation information. However, we find that 14 trusted signing certificates from 9 organizations fail to include revocation data in at least some of their certificates, and in 5 cases do not supply revocation data in any of their signed certificates.

2.7 Unexpected Observations

We observed a variety of unexpected phenomenon during our scans over the past year. We describe these observations here.

2.7.1 CA Certs with Multiple Parents

Of the 1,832 browser-trusted signing certificates we found, 380 shared their subject, public key, and subject key identifier with another browser-trusted certificate forming 136 groups of "sibling" CA certificates. Because of this, leaf certificates can have more than one parent from the browsers' point of view. We find that only 37.4% of browser trusted leaf

certificates have a single parent; 38.7% have two parents; 12.3% have three; 11.3% have four; and a small number have 5–9 valid parents. Depending on which parent is presented in a trust chain, the perceived validity of the leaf certificate can change. For example, if the presented intermediate certificate has expired, then the leaf certificate will be considered invalid. We note that subject key identified sometimes also specifies additional constraints such as a constraint on issuer serial number. However, we find that only a handful of certificates contain additional constraints.

In 86 of the 136 groups of sibling certificates, the signing certificates had differing validity periods. In four sets, one of the certificates was revoked, in a separate four sets, each authority was in a different browser or OS root store, and in 49 cases the authorities were signed by different parent authorities. While previous studies found evidence of this phenomenon, we were not aware of the prevalence of this behavior. We are not aware of any security vulnerabilities that are introduced by this practice, but we do find that 43,674 (1.35%) of the browser-trusted certificates are presented with the incorrect parent, which limits their perceived validity (e.g. the presented CA certificate expires earlier the leaf certificate, but another parent exists with a later expiration date).

2.7.2 CA Certs with Negative Path Lengths

We find that 1,395 browser-trusted CA certificates have a negative path length constraint, which renders them unable to sign any certificates due to a path length restriction earlier in the trust chain. These malformed intermediate certificates were signed by the Government of Korea and provided to educational institutions ranging from elementary schools to universities, libraries, and museums. However, because they are still technically CA certificates, web browsers including Mozilla Firefox and Google Chrome will not recognize them as valid leaf certificates.

We do not include these certificates when referring to the set of browser-trusted authorities because they are unable to sign any certificates and therefore do not have the

same influence as other valid authorities. However, we note that some less common client implementations may fail to properly check the path length constraint and incorrectly treat these as valid. One of these CA certificates, issued to a Korean elementary school, was compromised by Heninger [62], who factored the 512-bit key a few hours after the certificate expired.

2.7.3 Mis-issued CA Certificates

We found one mis-issued signing certificate during the course of our study, which was issued for `*.EGO.GOV.TR`, by Turktrust, a small Turkish certificate authority. We found the certificate served as a leaf certificate on what appeared to be an unconfigured IIS server on a Turkish IP address. We saw 487 certificates that were signed by Turktrust over the course of our study. All were for Turkish organizations or the Turkish Government; we saw no evidence of other mis-issued certificates.

The certificate was later found by Google after being used to sign a Google wildcard certificate [78] and was revoked by Turktrust on December 26, 2012. It was last seen in our scans on December 27, 2012.

2.7.4 Site Certificates with Invalid Domains

We find that 4.6% (149,902) of browser-trusted certificates contain a common name (CN) or subject alternate name for a locally scoped domain or private IP address. Because these names are not fully qualified, the intended resource is ambiguous and there is no identifiable owner. As such, these local domain names frequently appear on more than one certificate. In one example, there are 1,218 browser-trusted certificates for the domain `mail` owned by organizations ranging from the U.S. Department of Defense to the Lagunitas Brewing Company.

The vast majority of certificates appear to be related to mail services. Of the 157,861 certificates with locally scoped names, 25,964 contain the name `exchange` (Microsoft

Exchange Mail Server) and 99,773 contain a variation on the name `mail`. More than 100,000 of the certificates contain a domain ending in `.local`.

We suspect that certificates include these locally scoped names in order to facilitate users that are part of an Active Directory domain in connecting to their local Exchange mail server. In this scenario, the integrated DNS service in Active Directory will automatically resolve locally scoped names to the correct server on the domain. However, these clients will receive a name mismatch error if the TLS certificate presented by the Exchange Server does not match the locally scoped name that was originally resolved. Instead of requiring users to use the fully qualified domain name (FQDN) of the Exchange Server unlike other servers on the domain, certificate authorities include the local name of the Exchange server. In the case of certificates ending in `.local`, Active Directory Forests are generally rooted in an FQDN. In cases where organizations have not registered an FQDN for their forest, Active Directory elects to use the `.local` TLD.

Unfortunately, this practice does not provide security against man-in-the-middle attacks. It is trivial to procure a certificate with the same locally scoped name as another organization. Because there is no identifiable owner for the domain, both certificates are equally valid, and the subsequent certificate can be used to impersonate the original organization.

2.8 Adoption Trends

We observe a steady, linear increase in nearly all aspects of HTTPS adoption between June 2012 and April 2013, as shown in Figure 2.7. Most notably, there is a 23.0% increase in the number of Alexa Top 1 Million domains serving trusted certificates and a 10.9% increase in the number of unique browser-trusted certificates found during each scan. During this time, the Netcraft Web survey finds only a 2.2% increase in the number of active sites that respond over HTTP [99]. Based on the Netcraft Survey, we find an 8.5% increase in the number of websites utilizing HTTPS from 1.61% to 1.75%. This indicates that the increase in the number of certificates is not solely dependent on the growth of the Internet,

but that there is an increase in the adoption of HTTPS in existing sites. We also note a 16.8% increase in the number of extended validation certificates, a 19.6% increase in the number of hosts serving HTTPS on port 443, and an 11.1% increase in the total number of TLS certificates over this period.

The market share of each authority did not change drastically over the past year. In terms of number of valid signed leaf certifictes, Symantec grew 6%, GoDaddy 13%, and Comodo 17%. During this time, there was a 10.9% increase in the global number of unique valid browser-trusted certificates. StartCom, a smaller authority based in Israel that offers free basic certificates, grew by 32% over the course of the year, from 2.17% to 2.56% market share. We plot the growth of the top authorities in Figure 2.8.

2.9 Discussion

Analyzing the certificate authority ecosystem from a global perspective reveals several current practices that put the entire HTTPS ecosystem at risk. In this section, we discuss our observations and possible paths forward.

Ignoring Foundational Principles The security community has several widely accepted best practices such as the *principle of least privilege* and *defense in depth*. However, these guidelines are not being well applied within one of our most security critical ecosystems. For instance, there are several technical practices already at our disposal for limiting the scope of a signing certificate, including setting name or path length constraints and distributing leaf certificates among a large number intermediate certificates. There are clear cases for using these restrictions, but the vast majority of the time, CAs are not fully utilizing these options.

One example of how defense in depth successfully prevented compromise can be seen in the 1,400 signing certificates that were mis-issued to organizations in South Korea (Section 2.7.2). In this case, a path length constraint on a grandparent certificate prevented

this error from becoming a massive vulnerability. To put this in context, if *defense in depth* had not been practiced, the erroneous action of a single certificate authority would have tripled the number of organizations controlling a valid signing certificate overnight. Unfortunately, while a path length constraint was in place for this particular situation, more than 40% of CA certificates do not have any constraints in place to prevent this type of error and only a small handful use name constraints.

In a less fortunate example, Turktrust accidentally issued a signing certificate to one of its customers that ultimately signed a valid certificate for `*.google.com` (Section 2.7.3). If name or path constraints had been applied to Turktrust's CA intermediate certificate, the incident could have been avoided or, at the very least, reduced in scope. In other situations, the risk associated with compromise of a single signing certificate could be decreased by simply spreading issuance across multiple certificates (Section 2.5.3).

Standards and Working Groups The CA/Browser Forum is a voluntary working group composed of certificate authorities and Internet browser software vendors. The group has recently attempted to resolve many of the security risks previously introduced by certificate authorities, and in November 2011, they adopted guidelines for certificate authorities [31] that touch on many of the concerns we raise.

However, with only 20% of the organizations controlling signing certificates being commercial certificate authorities and less than 25% of commercial authorities participating in the workgroup, there remains a disconnect. It is unclear how many organizations are aware of the existence of the baseline standard, but it is clear that a large number of organizations are either unaware or are choosing to ignore the forum's baseline requirements. One example of this non-adherence can be seen in the agreement to cease the issuance of certificates containing internal server names and reserved IP addresses. Despite the ratification of this policy, more than 500 certificates containing internal server names and which expire after November 1, 2015 have been issued since July, 1, 2012 by CA/B Forum members

(Section 2.7.4).

Without any enforcement, members of the CA/Browser Forum have disregarded adopted policies and we expect that other organizations are unaware of the standards. There is still work required from the security community to reign in these additional authorities and to follow up with members that are disregarding existing policies.

Browsers to Lead the Way Web browser and operating system maintainers are in a unique position to set expectations for certificate authorities, and it is encouraging to see increasing dialogue in the CA/Browser Forum. However, browsers also have a responsibility to commit resources towards a healthier ecosystem. Many new, more secure technologies are dependent on support in common browsers and web servers. Without browser compatibility, certificate authorities lack incentive to adopt new, more secure options regardless of support from the security community.

This can immediately be seen in the deployment of name constraints. We find that the vast majority of the CA certificates issued to non-CAs are used to issue certificates to a small number of domains and, as such, could appropriately be scoped using name constraints with little impact on day-to-day operations. Restricted scopes have been shown to greatly reduce the attack surface of the CA ecosystem [74], and with 80% of existing signing certificates belonging to organizations other than commercial certificate authorities, there is a clear and present need for name constraints (Section 2.5). However, Safari and Google Chrome on Mac OS do not currently support the critical server name constraint extension. As a result, any certificate signed using an appropriately scoped CA certificate with the extension marked as critical will be rejected on these platforms. Therefore, while there is community consensus on the value of server name constraints, progress will be slow until all browsers support the extension.

Failing to Recognize Cryptographic Reality It is encouraging to find that over 95% of trusted leaf certificates and 95% of trusted signing certificates use NIST recommended key

sizes [25]. However, more than 50 root authorities continue to use 1024-bit RSA keys, the last of which expires in 2040—more than 20 years past recommended use for a key of this size (Section 2.5.7). Authorities are not adequately considering long-term consequences of authority certificates and need to anticipate what the cryptographic landscape will be in the future. Many of these root certificates were signed prior to guidelines against such long-lived CA certificates. However, today, we need to be working to resolve these past errors and preparing to remove now-inappropriate root CAs from browser root stores.

2.10 Conclusion

In this work, we completed the largest known measurement study of the HTTPS certificate ecosystem by performing 110 comprehensive scans of the IPv4 HTTPS ecosystem over a 14 month period. We investigated the organizations that the HTTPS ecosystem depends on and identified several specific practices employed by certificate authorities that lead to a weakened public key infrastructure. We provided updated metrics on many aspects of HTTPS and certificate deployment along with adoption trends over the last year. Lastly, we discussed the high-level implications of our results and make several recommendations for strengthening the ecosystem. Our study shows that regular active scans provide detailed and temporally fine-grained visibility into this otherwise opaque area of security critical infrastructure. We are publishing the data from our scans at https://scans.io/ in the hope that it will assist other researcher in further investigating the HTTPS ecosystem.

Organization Type	Organizations		Authorities		Leaf Certificates		Hosts	
Academic Institution	273	(39.79%)	292	(15.93%)	85,277	(2.46%)	85,277	(0.92%)
Commercial CA	135	(19.67%)	819	(44.70%)	3,260,454	(94.20%)	3,260,454	(76.33%)
Government Agency	85	(12.39%)	250	(13.64%)	17,865	(0.51%)	17,865	(0.23%)
Corporation	83	(12.09%)	191	(10.42%)	30,115	(0.87%)	30,115	(4.80%)
ISP	30	(4.37%)	58	(3.16%)	8,126	(0.23%)	8,126	(1.55%)
IT/Security Consultant	29	(4.22%)	88	(4.80%)	22,568	(0.65%)	22,568	(0.98%)
Financial Institution	17	(2.47%)	49	(2.67%)	2,412	(0.06%)	2,412	(0.03%)
Unknown	*unknown*		15	(0.81%)	2,535	(0.07%)	2,535	(0.02%)
Hosting Provider	7	(1.02%)	12	(0.65%)	10,598	(0.30%)	10,598	(14.70%)
Nonprofit Org	7	(1.02%)	15	(0.81%)	11,480	(0.33%)	11,480	(0.11%)
Library	5	(0.72%)	6	(0.32%)	281	(0.00%)	281	(0.00%)
Museum	4	(0.58%)	4	(0.21%)	35	(0.00%)	35	(0.00%)
Healthcare Provider	3	(0.43%)	4	(0.21%)	173	(0.00%)	173	(0.00%)
Religious Institution	1	(0.14%)	1	(0.05%)	11	(0.00%)	11	(0.00%)
Military	1	(0.14%)	27	(1.47%)	9,017	(0.26%)	9,017	(0.27%)

Table 2.3: **Types of Organizations with Signing Certificates** — We found 1,832 valid browser-trusted signing certificates belonging to 683 organizations. We classified these organizations and find that more than 80% of the organizations that control a signing certificate are not commercial certificate authorities.

Parent Company	Signed Leaf Certificates	
Symantec	1,184,723	(34.23%)
GoDaddy.com	1,008,226	(29.13%)
Comodo	422,066	(12.19%)
GlobalSign	170, 006	(4.90%)
DigiCert Inc	145,232	(4.19%)
StartCom Ltd.	88,729	(2.56%)
Entrust, Inc.	76,990	(2.22%)
Network Solutions	62,667	(1.81%)
TERENA	42,310	(1.22%)
Verizon Business	32,127	(0.92%)

Table 2.4: **Top Parent Companies** — Major players such as Symantec, GoDaddy, and Comodo have acquired smaller CAs, leading to the 5 largest companies issuing 84.6% of all trusted certificates.

Organization	Signed Leaf Certificates	
GoDaddy.com, Inc.	913,416	(28.6%)
GeoTrust Inc.	586,376	(18.4%)
Comodo CA Limited	374,769	(11.8%)
VeriSign, Inc.	317,934	(10.0%)
Thawte, Inc.	228,779	(7.2%)
DigiCert Inc	145,232	(4.6%)
GlobalSign	117,685	(3.7%)
Starfield Technologies	94,794	(3.0%)
StartCom Ltd.	88,729	(2.8%)
Entrust, Inc.	76929	(2.4%)

Table 2.5: **Top Certificate Authorities** — The top 10 commercial certificate authorities control 92.4% of trusted certificates present in our March 22, 2013 scan.

Systems Valid In	Roots	CAs	Signed
Windows Only	125	283	24,873
Mozilla Only	2	3	23
Apple Only	26	30	3,410
Windows & Mozilla	32	97	12,282
Windows & Apple	31	47	9,963
Mozilla & Apple	3	3	0
All Browsers	109	1,346	8,945,241

Table 2.6: **Differences in Browser and OS Root Stores** — While there are significant differences in the root certificates stores, 99.4% of trusted certificates are trusted in all major browsers.

Type	Root Authorities		Recursively Signed	
ECDSA	6	(1.8%)	0	(0%)
RSA (1024-bit)	53	(16.0%)	1,694,526	(48.6%)
RSA (2028-bit)	202	(61.0%)	1,686,814	(48.4%)
RSA (4096-bit)	70	(21.2%)	102,139	(2.9%)

Table 2.7: **Key Distribution for Trusted Roots** — The distribution of keys for root certificates shipped with major browsers and OSes.

Key Type	Authorities		Signed Leaves	
ECDSA	6	(0.3%)	0	(0%)
RSA (1024-bit)	134	(7.3%)	133,391	(4.2%)
RSA (2048-bit)	1,493	(78.9%)	3,034,751	(95.3%)
RSA (4096-bit)	198	(10.5%)	16,969	(0.5%)

Table 2.8: **Key Distribution for Trusted Signing Certificates**

Key Type	All Trusted		Valid Trusted
RSA ($\leq$ 512-bit)	2,631	(0.1%)	16
RSA (768-bit)	73	(0.0%)	0
RSA (1024-bit)	341,091	(10.5%)	165,637
RSA (1032–2040-bit)	23,888	(0.7%)	105
RSA (2048-bit)	2,816,757	(86.4%)	2,545,693
RSA (2056–4088-bit)	1,006	(0.0%)	921
RSA (4096-bit)	74,014	(2.3%)	65,780
RSA ($>$ 4096-bit)	234	(0.0%)	192
DSA (all)	17	(0.0%)	7
ECDSA (all)	0	(0.0%)	0

Table 2.9: **Trusted Leaf Certificate Public Key Distribution**

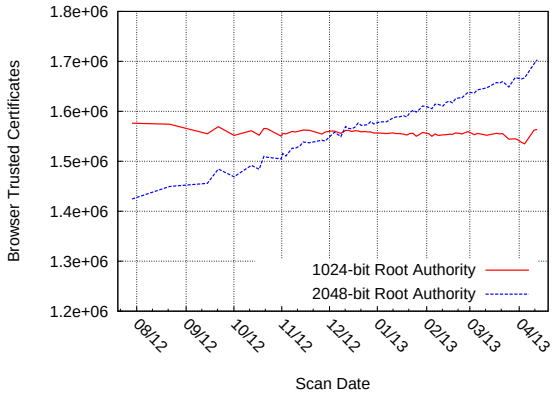

Figure 2.4: **Temporal Trends in Root Key Size** — We find that 48.7% of browser-trusted leaf certificates are dependent on 1024-bit RSA based root authorities, contrary to recommended practice [24].

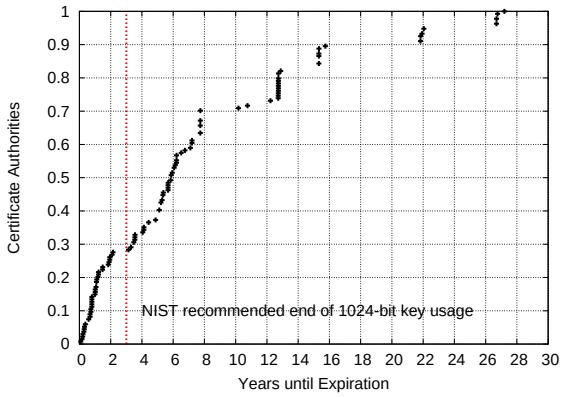

Figure 2.5: **Expiration of 1024-bit Root Certificates** — This figure shows when trusted 1024-bit RSA CA certificates expire. We note that more than 70% expire after 2016 when NIST recommends discontinuing the use of 1024-bit keys.

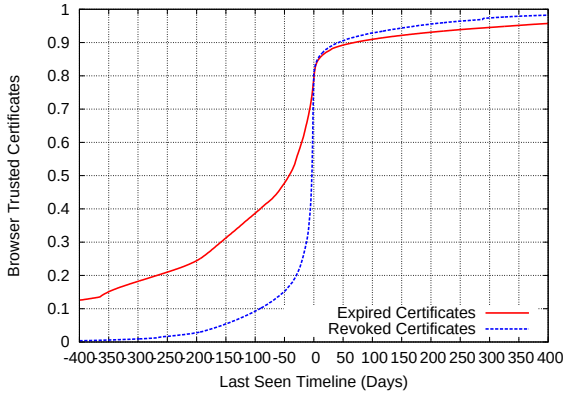

Figure 2.6: **CDF of Certificate Removal** — We find that 20% of expiring certificates and 19.5% of revoked certificates are removed retroactively (to the right of 0 days).

Type	Trusted Certificates	
SHA-1 with RSA Encryption	5,972,001	(98.7%)
MD5 with RSA Encryption	32,905	(0.54%)
SHA-256 with RSA Encryption	15,297	(0.25%)
SHA-512 with RSA Encryption	7	(0.00%)
MD2 with RSA Encryption	21	(0.00%)
Other	29,705	(0.49%)

Table 2.10: **Trusted Leaf Certificate Signature Algorithms**

Status	Hosts	
Expired	595,168	(5.80%)
Not Yet Valid	1,966	(0.02%)
Revoked	28,033	(0.27%)
No Trust Chain	654,667	(6.30%)
Misordered Chain	25,667	(0.24%)
Incorrect Chain	11,761	(0.14%)
Unnecessary Root	4,365,321	(42.2%)
Optimally Configured	4,657,133	(45.0%)

Table 2.11: **Common Server Certificate Problems** — We evaluate hosts serving browser-trusted certificates and classify common certificate and server configuration errors. The number of misconfigured hosts indicates that procuring certificates and correctly configuring them on servers remains a challenge for many users.

Revocation Reason	Revoked Certificates	
Cessation Of Operation	101,370	(64.9%)
Not Provided	31514	(20.2%)
Affiliation Changed	7,384	(4.7%)
Privilege Withdrawn	5,525	(3.5%)
Unspecified	4,523	(2.9%)
Superseded	3,887	(2.5%)
Key Compromise	1,945	(1.2%)
Certificate Hold	45	(0.0%)
CA Compromise	2	(0.0%)
Total	**156,195**	

Table 2.12: **Reasons for Revocation** — We find that 10,220 (2.5%) of the browser trusted certificates seen in our study were eventually revoked. Both of the "CA Compromised" revocations were due to the DigiNotar compromise [29].

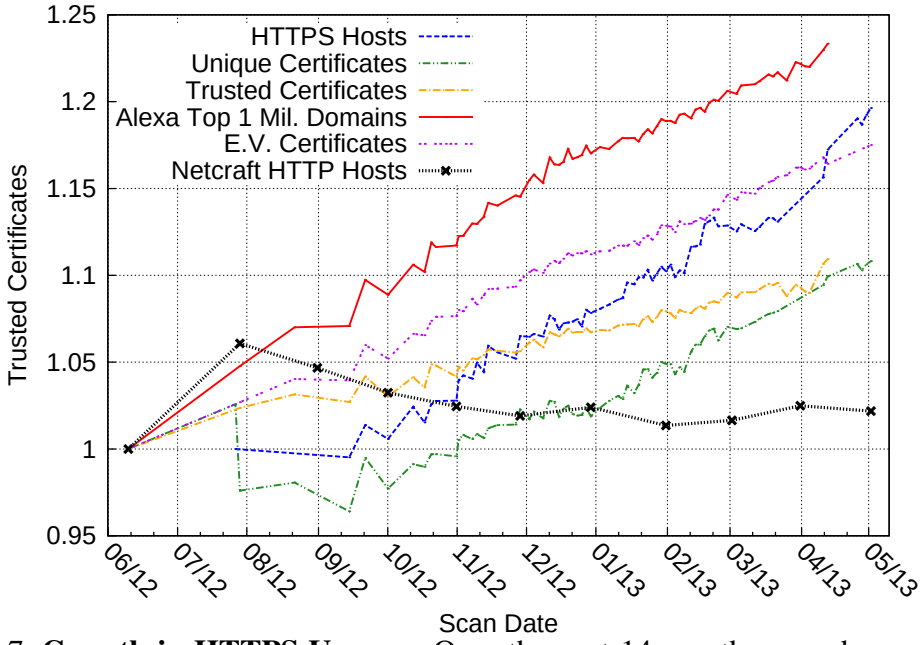

Figure 2.7: **Growth in HTTPS Usage** — Over the past 14 months, we observe between 10-25% growth of all aspects of HTTPS usage.

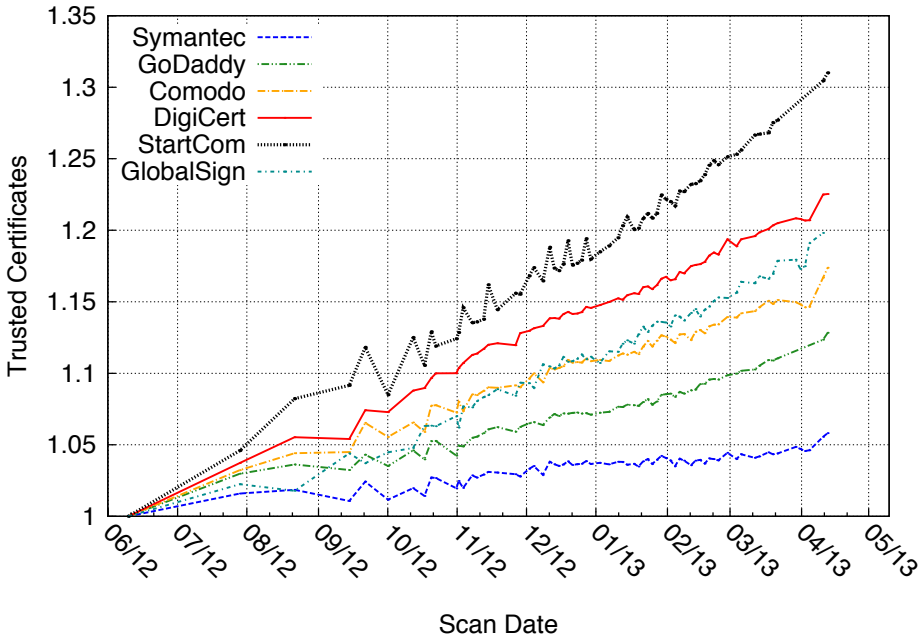

Figure 2.8: **Change in Authority Market Share** — In this figure, we show the individual growth of the top 10 most prolific certificate authorities.

CHAPTER III

The Matter of Heartbleed [42]

3.1 Introduction

In March 2014, researchers found a catastrophic vulnerability in OpenSSL, the cryptographic library used to secure connections in popular server products including Apache and Nginx. While OpenSSL has had several notable security issues during its 16 year history, this flaw—the *Heartbleed* vulnerability—was one of the most impactful. Heartbleed allows attackers to read sensitive memory from vulnerable servers, potentially including cryptographic keys, login credentials, and other private data. Exacerbating its severity, the bug is simple to understand and exploit.

In this work, we analyze the impact of the vulnerability and track the server operator community's responses. Using extensive active scanning, we assess who was vulnerable, characterizing Heartbleed's scope across popular HTTPS websites and the full IPv4 address space. We also survey the range of protocols and server products affected. We estimate that 24–55% of HTTPS servers in the Alexa Top 1 Million were initially vulnerable, including 44 of the Alexa Top 100. Two days after disclosure, we observed that 11% of HTTPS sites in the Alexa Top 1 Million remained vulnerable, as did 6% of all HTTPS servers in the public IPv4 address space. We find that vulnerable hosts were not randomly distributed, with more than 50% located in only 10 ASes that do not reflect the ASes with the most HTTPS hosts. In our scans of the IPv4 address space, we identify over 70 models of vulnerable embedded

devices and software packages. We also observe that both SMTP+TLS and Tor were heavily affected; more than half of all Tor nodes were vulnerable in the days following disclosure.

Our investigation of the operator community's response finds that within the first 24 hours, all but 5 of the Alexa Top 100 sites were patched, and within 48 hours, all of the vulnerable hosts in the top 500 were patched. While popular sites responded quickly, we observe that patching plateaued after about two weeks, and 3% of HTTPS sites in the Alexa Top 1 Million remained vulnerable almost two months after disclosure.

In addition to patching, many sites replaced their TLS certificates due to the possibility that the private keys could have been leaked and is the focus of this thesis. We analyze certificate replacement and find that while many of the most popular websites reacted quickly, less than a quarter of Alexa Top 1 Million sites replaced certificates in the week following disclosure. Even more worryingly, only 10% of the sites that were vulnerable 48 hours after disclosure replaced their certificates within the next month, and of those that did, 14% neglected to change the private key, gaining no protection from certificate replacement.

Finally, starting three weeks after disclosure, we undertook a large-scale notification effort and contacted the operators responsible for the remaining vulnerable servers. By contacting the operators in two waves, we could conduct a controlled experiment and measure the impact of notification on patching. We report the effects of our notifications, observing a surprisingly high 47% increase in patching by notified operators.

We draw upon these observations to discuss both what went well and what went poorly in the aftermath of Heartbleed. By better understanding the lessons of this security disaster, the technical community can respond more effectively to such events in the future.

1

[1]This chapter is an excerpt from "The Matter of Heartbleed" [42]. I specifically collected and analyzed the data from the certificate replacement and revocation sections and helped write the sections included.

3.2 Background

On April 7, 2014, the OpenSSL project publicly disclosed the Heartbleed vulnerability, a bug in their implementation of the TLS Heartbeat Extension. The vulnerability allowed attackers to remotely dump protected memory—including data passed over the secure channel and private cryptographic keys—from both clients and servers. In this section, we provide a brief history of OpenSSL, the Heartbeat Extension, and details of the vulnerability and its disclosure.

3.2.1 OpenSSL: A Brief History

OpenSSL is a popular open-source cryptographic library that implements the SSL and TLS protocols. It is widely used by server software to facilitate secure connections for web, email, VPN, and messaging services. The project started in 1998 and began tracking vulnerabilities in April 2001.

Over the last 13 years, OpenSSL has documented six code execution vulnerabilities that allowed attackers to compromise private server data (e.g., private cryptographic keys and messages in memory) and execute arbitrary code. The project has faced eight information leak vulnerabilities, four of which allowed attackers to retrieve plaintext, and two of which exposed private keys. Two of the vulnerabilities arose due to protocol weaknesses; the remainder came from implementation errors.

The Heartbleed bug reflects one of the most impactful vulnerabilities during OpenSSL's history for several reasons: (1) it allowed attackers to retrieve private cryptographic keys and private user data, (2) it was easy to exploit, and (3) HTTPS and other TLS services have become increasingly popular, resulting in more affected services.

3.2.2 TLS Heartbeat Extension

The Heartbeat Extension allows either end-point of a TLS connection to detect whether its peer is still present, and was motivated by the need for session management in Datagram

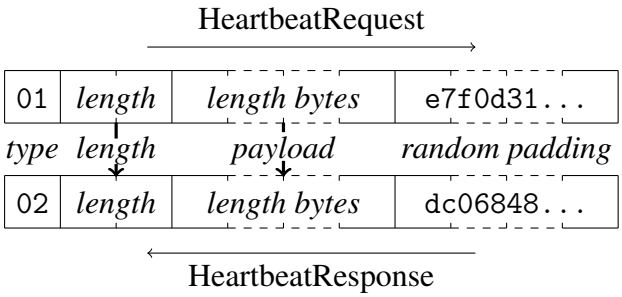

Figure 3.1: **Heartbeat Protocol.** Heartbeat requests include user data and random padding. The receiving peer responds by echoing back the data in the initial request along with its own padding.

TLS (DTLS). Standard implementations of TLS do not require the extension as they can rely on TCP for equivalent session management.

Peers indicate support for the extension during the initial TLS handshake. Following negotiation, either end-point can send a `HeartbeatRequest` message to verify connectivity. The extension was introduced in February 2012 in RFC 6520 [117], added to OpenSSL on December 31, 2011, and released in OpenSSL Version 1.0.1 on March 14, 2012.

`HeartbeatRequest` messages consist of a one-byte type field, a two-byte payload length field, a payload, and at least 16 bytes of random padding. Upon receipt of the request, the receiving end-point responds with a similar `HeartbeatResponse` message, in which it echoes back the `HeartbeatRequest` payload and its own random padding, per Figure 3.1.

3.2.3 Heartbleed Vulnerability

The OpenSSL implementation of the Heartbeat Extension contained a vulnerability that allowed either end-point to read data following the payload message in its peer's memory by specifying a payload length larger than the amount of data in the `HeartbeatRequest` message. Because the payload length field is two bytes, the peer responds with up to 2^{16} bytes (˜64 KB) of memory. The bug itself is simple: the peer trusts the attacker-specified length of an attacker-controlled message.

The OpenSSL patch adds a bounds check that discards the `HeartbeatRequest` message

Date	Event
03/21	Neel Mehta of Google discovers Heartbleed
03/21	Google patches OpenSSL on their servers
03/31	CloudFlare is privately notified and patches
04/01	Google notifies the OpenSSL core team
04/02	Codenomicon independently discovers Heartbleed
04/03	Codenomicon informs NCSC-FI
04/04	Akamai is privately notified and patches
04/05	Codenomicon purchases the `heartbleed.com` domain
04/06	OpenSSL notifies several Linux distributions
04/07	NCSC-FI notifies OpenSSL core team
04/07	OpenSSL releases version 1.0.1g and a security advisory
04/07	CloudFlare and Codenomicon disclose on Twitter
04/08	Al-Bassam scans the Alexa Top 10,000
04/09	University of Michigan begins scanning

Table 3.1: **Timeline of Events in March and April 2014.** The discovery of Heartbleed was originally kept private by Google as part of responsible disclosure efforts. News of the bug spread privately among inner tech circles. However, after Codenomicon independently discovered the bug and began separate disclosure processes, the news rapidly became public [61, 107].

if the payload length field exceeds the length of the payload. However, while the bug is easy to conceptualize and the fix is straight-forward, the potential impact of the bug is severe: it allows an attacker to read private memory, potentially including information transferred over the secure channel and cryptographic secrets [51, 109, 122].

3.2.4 Heartbleed Timeline

The Heartbleed vulnerability was originally found by Neel Mehta, a Google computer security employee, in March 2014 [61]. Upon finding the bug and patching its servers, Google notified the core OpenSSL team on April 1. Independently, a security consulting firm, Codenomicon, found the vulnerability on April 2, and reported it to National Cyber Security Centre Finland (NCSC-FI). After receiving notification that two groups independently discovered the vulnerability, the OpenSSL core team decided to release a patched version.

The public disclosure of Heartbleed started on April 7, 2014 at 17:49 UTC with the version 1.0.1g release announcement [107], followed by the public security advisory [106] released at 20:37 UTC; both announcements were sent to the OpenSSL mailing list. Several parties knew of the vulnerability in advance, including CloudFlare, Akamai and Facebook. Red Hat, SuSE, Debian, FreeBSD and ALT Linux were notified less than 24 hours before the public disclosure [61]. Others, such as Ubuntu, Gentoo, Chromium, Cisco, and Juniper were not aware of the bug prior to its public release. We present a timeline of events in Table 3.1.

3.3 Patching

In this section, we discuss the patching behavior that occurred subsequent to the disclosure.

3.3.1 Popular Websites

Popular websites did well at patching. As mentioned above, only five sites in the Alexa Top 100 remained vulnerable when Al-Bassam completed his scan 22 hours after disclosure. All of the top 100 sites were patched by the time we started our scans, 48 hours after disclosure. Our first scan of the Alexa Top 1 Million found that 11.5% of HTTPS sites remained vulnerable. The most popular site that remained vulnerable was `mpnrs.com`, ranked 689th globally and 27th in Germany. Similarly, all but seven of the vulnerable top 100 sites replaced their certificate in the first couple of weeks following disclosure. Most interestingly, `godaddy.com`, operator of the largest commercial certificate authority, did not change their certificates until much later. The other six sites are `mail.ru`, `instagram.com`, `vk.com`, `sohu.com`, `adobe.com`, and `kickass.to`.

As shown in Figure 3.2, while many Alexa Top 1 Million sites patched within the first week, the patch rate quickly dropped after two weeks, with only a very modest decline between April 26 and June 4, 2014. While top sites in North America and Europe were

initially more vulnerable than Asia and Oceania (presumably due to more prevalent use of OpenSSL-based servers), they all followed the same general patching pattern visible in Figure 3.2.

3.3.2 Internet-Wide HTTPS

As can be seen in Figure 3.2, the patching trend for the entire IPv4 address space differs from that of the Alexa Top 1 Million. The largest discrepancy occurs between April 22, 14:35 EDT and April 23, 14:35 EDT, during which the total number of vulnerable hosts fell by nearly a factor of two, from 4.6% to 2.6%. This dropoff occurred because several heavily affected ASes patched many of their systems within a short time frame. The shift from 4.6% to 3.8% between 14:35 and 22:35 on April 22 reflects Minotavar Computers patching 29% of their systems, ZeXoTeK IT-Services patching 60%, and Euclid Systems patching 43%. Between April 22, 22:35 and April 23, 06:35, Minotavar Computers continued to patch systems. The last major drop from 3.4% to 2.6% (06:35–14:35 on April 23) was primarily due to Minotavar Computers patched remaining systems, and to a lesser extent, Euclid Systems and Vivid Hosting.

3.3.3 Comparison to Debian Weak Keys

In 2008, a bug was discovered in the Debian OpenSSL package, in which the generation of cryptographic keys had a severely limited source of entropy, reducing the space of possible keys to a few hundred thousand. The lack of entropy allowed attackers to fully enumerate the SSL and SSH keys generated on Debian systems, thus making it vital for Debian OpenSSL users to generate fresh replacement keys.

Yilek et al. measured the impact of the vulnerability and patching behavior by performing daily scans of HTTPS servers [132]. Given the similarities in the severity and nature of remediation between this event and Heartbleed, we compared the community's responses to both disclosures.

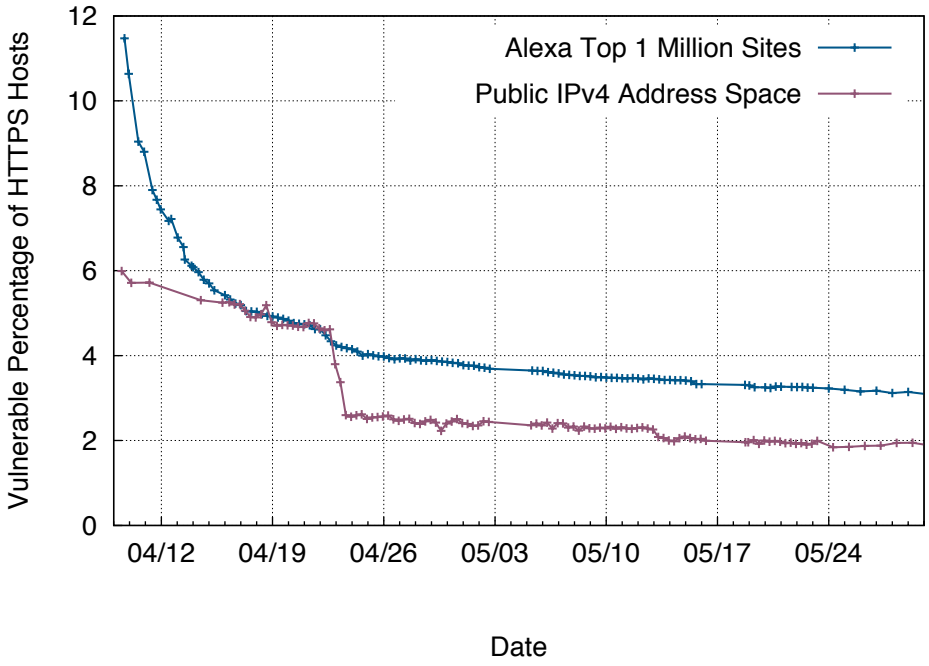

Figure 3.2: **HTTPS Patch Rate.** We track vulnerable web servers in the Alexa Top 1 Million and the public IPv4 address space. We track the latter by scanning independent 1% samples of the public IPv4 address space every 8 hours. Between April 9 and June 4, the vulnerable population of the Alexa Top 1 Million shrank from 11.5% to 3.1%, and for all HTTPS hosts from 6.0% to 1.9%.

A key methodological issue with conducting such a comparison concerns ensuring we use an "apples-to-apples" metric for assessing the extent of the community's response to each event. The comparison is further complicated by the fact that our Heartbleed measurements sample a different 1% of the Internet each scan. We do the comparison by framing the basic unit of "did an affected party respond" in terms of aggregate entities very likely controlled by the same party (and thus will update at the same time). To do so, we define an entity as a group of servers that all present the same certificate during a particular measurement. This has the potential for fragmenting groups that have partially replaced their certificates, but we argue that this effect is likely negligible since the number of entities stays roughly constant across our measurements. Note that this definition of entity differs from the "host-cert" unit used in [132], in which groups were tracked as a whole from the first measurement.

Figure 3.3 shows for both datasets the fraction of unfixed entities to the total number of

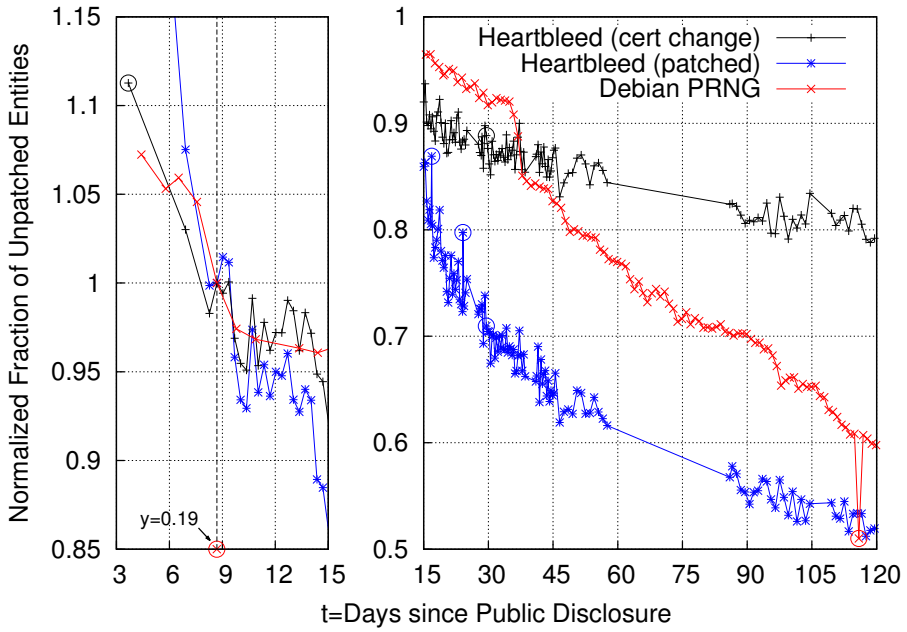

Figure 3.3: **Comparison of the Patch Rates of the Debian PRNG and Heartbleed Vulnerabilities.** The y-axis is normalized at 8.7 days, indicated by the vertical striped line. Thus, the fraction of unpatched entities at a given time is relative to the fraction at 8.7 days after disclosure, for each dataset. Except for the points marked by ○, for each measurement the size of the Debian PRNG entity population was $n = 41{,}200 \pm 2{,}000$, and for Heartbleed, $n = 100{,}900 \pm 7{,}500$. Due to a misconfiguration in our measurement setup, no Heartbleed data is available days 58–85.

entities per measurement. We consider an entity as "fixed" in the Debian PRNG dataset if the certificate now has a strong public key (and previously did not), otherwise "unfixed". For our Heartbleed IPv4 dataset (labelled "patch"), we deem an entity as "fixed" if *all* servers presenting the same certificate are now no longer vulnerable, "unfixed" otherwise.

This data shows that entities vulnerable to Heartbleed patched somewhat more quickly than in the Debian scenario, and continue to do so at a faster rate. It would appear that aspects of the disclosure and publicity of Heartbleed did indeed help with motivating patching, although the exact causes are difficult to determine.

Note that for the Debian event, it was very clear that affected sites had to not only patch but to also issue new certificates, because there was no question that the previous certificates were compromised. For Heartbleed, the latter step of issuing new certificates was not as

pressing, because the previous certificates were compromised only if attackers had employed the attack against the site prior to patching *and* the attack indeed yielded the certificate's private key.

Given this distinction, we also measured whether entities changed their certificates after patching Heartbleed.[2] To do so, we now define an entity as a group of servers that all present the same certificate during both a particular measurement and all previous measurements. We regard an entity as "unfixed" if *any* server presenting that certificate is vulnerable at any time during this time frame and "fixed" otherwise. Again, we argue that fragmentation due to groups having their servers only been partially patched is likely negligible. We label this data as "cert change" in Figure 3.3. We see that while entities patched Heartbleed faster than the Debian PRNG bug, they replaced certificates more slowly, which we speculate reflects a perception that the less-definitive risk of certificate compromise led a number of entities to forgo the work that reissuing entails.

3.4 Certificate Ecosystem

Heartbleed allowed attackers to extract private cryptographic keys [122]. As such, the security community recommended that administrators generate new cryptographic keys and revoke compromised certificates [58]. In this section, we analyze to what degree operators followed these recommendations.

3.4.1 Certificate Replacement

To track which sites replaced certificates and cryptographic keys, we combined data from our Heartbleed scans, Michigan's daily scans of the HTTPS ecosystem [43], and ICSI's Certificate Notary service [22]. Of the Alexa sites we found vulnerable on April 9 (2 days after disclosure), only 10.1% replaced their certificates in the month following disclosure (Figure 3.4). For comparison, we observed that 73% of vulnerable hosts detected on April 9

[2] See Section 3.4.1 for a discussion on the replacement of public/private key pairs in addition to certificates.

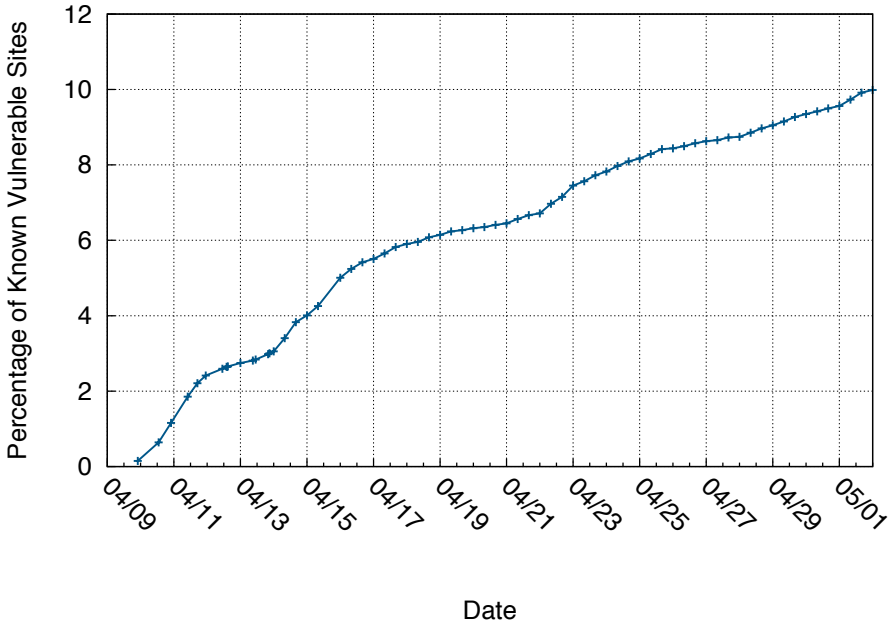

Figure 3.4: **Certificate Replacement on Vulnerable Alexa Sites.** We monitored certificate replacement on vulnerable Alexa Top 1 Million sites and observe only 10% replaced certificates in the month following public disclosure.

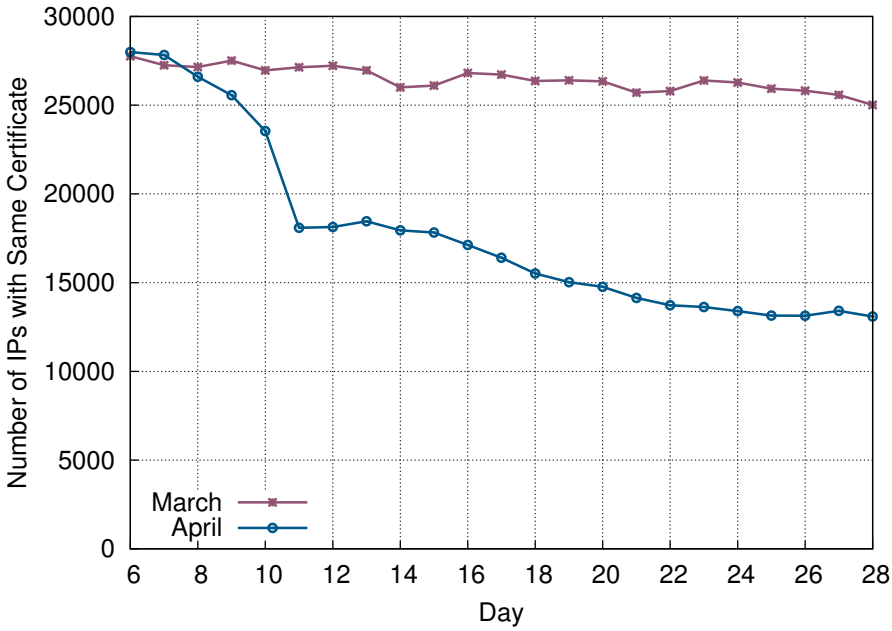

Figure 3.5: **ICSI Notary Certificate Changes.** Over both March and April, we track the number of servers who have the same certificate as on the 6th of each month. We only include servers that served the same certificate for the previous month.

patched in that same time frame, indicating that most hosts who patched did not replace certificates. In addition, it is striking to observe that only 19% of the vulnerable sites that did replace their certificates also revoked the original certificate in the same time frame, and even more striking that 14% *re-used the same private key*, thus gaining no actual protection by the replacement.

We find that 23% of all HTTPS sites in the Alexa Top 1 Million replaced certificates and 4% revoked their certificates between April 9 and April 30, 2014. While it may seem inverted that fewer vulnerable sites changed their certificates compared to all HTTPS sites in the Alexa Top 1 Million, our first scan was two days after initial disclosure. We expect that diligent network operators both patched their systems and replaced certificates within the first 48 hours post disclosure, prior to our first scan.

The ICSI Certificate Notary provides another perspective on changes in the certificate ecosystem, namely in terms of Heartbleed's impact on the sites that its set of users visit during their routine Internet use. In Figure 3.5, we show the difference in certificate replacement between March and April 2014. For the first four days after public disclosure on April 7, we observed a large drop in the number of servers with the same certificate as on April 6, indicating a spike in new certificates. After that, certificate changes progressed slowly yet steadily for the rest of the month. This matches our expectations that a number of operators patched their systems prior to our scans and immediately replaced certificates.

Ultimately, while popular websites did well at patching the actual vulnerability, a significantly smaller number replaced their certificates, and an even smaller number revoked their vulnerable certificates.

3.4.2 Certificate Revocation

When a certificate or key can no longer be trusted, sites can request the issuing CA to *revoke* the certificate. CAs accomplish this by publishing certificate revocation lists (CRLs) and supporting the Online Certificate Status Protocol (OCSP) for live queries. Even

though most vulnerable hosts failed to revoke old certificates, we observed about as many revocations in the three months following public disclosure as in the three previous years.

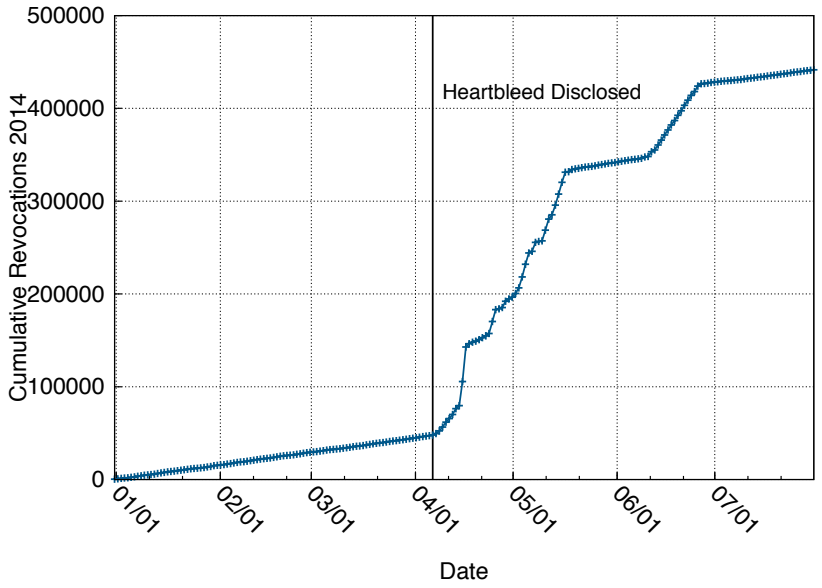

Figure 3.6: **Revocations after Heartbleed.** Certificate revocations dramatically increased after the disclosure. The jumps reflect GlobalSign (first) and GoDaddy (rest). However, only 4% of HTTPS sites in the Alexa Top 1 Million revoked their certificates between April 9 and April 30, 2014.

Prior to the vulnerability disclosure, we saw on average 491 (σ=101) revocations per day for certificates found in our scans. As seen in Figure 3.6, in the days following disclosure, the number of revocations dramatically increased. The sudden increases were due to individual CAs invalidating large portions of their certificates. Most notably, GlobalSign revoked 56,353 certificates over two days (50.2% of their visible certificates), and GoDaddy, the largest CA, revoked 243,823 certificates in week-long bursts over the following three months. GlobalSign's large number of revocations were precipitated by a major customer, CloudFlare, revoking all of their customers' certificates [110].

Revoking such a large number of certificates burdens both clients and servers. Clients must download large CRLs, which CAs must host. GlobalSign's CRL expanded the most,

from 2 KB to 4.7 MB due to CloudFlare revoking all of their certificates. CloudFlare hesitated to revoke their certificates, citing its significant cost, which they estimated would require an additional 40 Gbps of sustained traffic, corresponding to approximately $400,000 per month [110].

StartCom, a CA that offers free SSL certificates, came under fire for continuing their policy of charging for revocation after the Heartbleed disclosure [92]. However, revocation places a sizable financial strain on CAs due to bandwidth costs [13, 110].

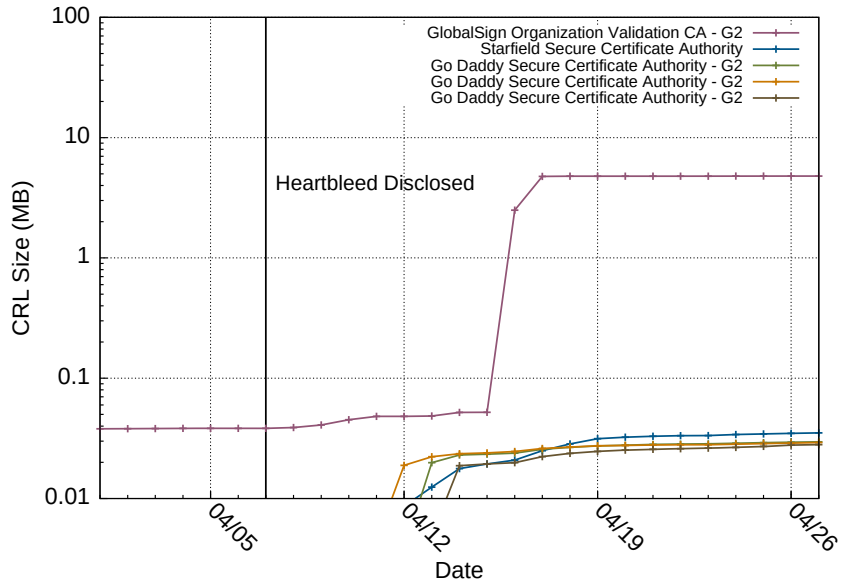

Figure 3.7: **Affected CRLs.** The CRLs with the greatest expansion during April 2014. Note that *Go Daddy Secure Certificate Authority–G2* uses 51 different CRLs, presumably limiting the size of each to avoid large CRL downloads caused by popular cryptographic libraries [80].

Unsurprisingly, CAs actively try to limit revocation costs. There has been a recent push in browsers to eliminate CRL usage in favor of OCSP and other custom techniques. Neither Firefox nor Chrome perform CRL checks for any certificates by default anymore. Internet Explorer does, however, with CAPI, the Windows certificate validation library. CAPI will only download and cache a CRL if 50 OCSP responses have been cached for a particular CA.

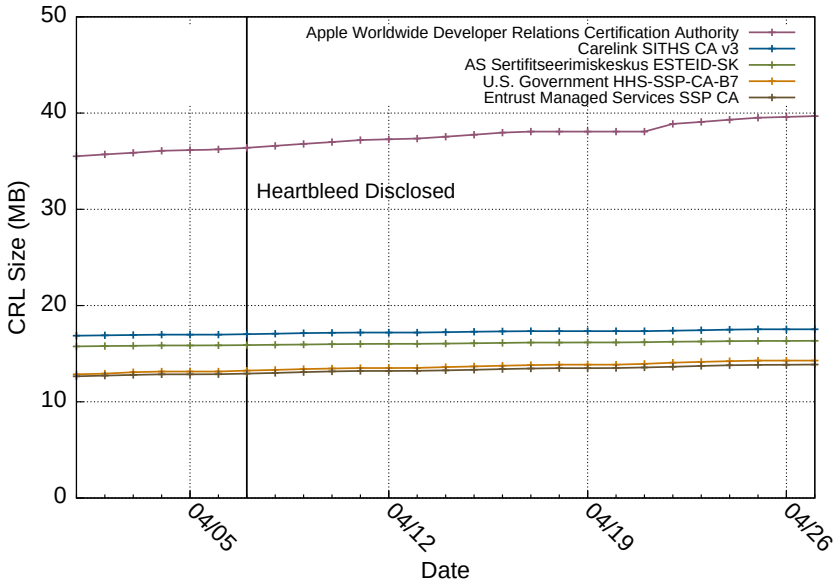

Figure 3.8: **Large CRL Growth.** The largest CRLs were unaffected by the disclosure. Most of the revoked certificates have not been seen during our scans of the Internet and the CRLs are not from large commercial CAs

CAs are aware of this practice and attempt to curtail the cost of revocation bandwidth further by splitting up their revocations into many distinct files. When the CRLs are downloaded from a particular certificate, they contain only a fraction of the CA's total revocation list. Although this diminishes the security afforded by downloading the CRL, it saves the CA money [80].

Interestingly, the largest CRLs did not increase in size dramatically after the disclosure. Figure 3.8 shows the expansion of the five largest CRLs. All five of the CRLs follow a linear growth pattern and appear largely unaffected by the disclosure. This is in stark contrast to Figure 3.7 which shows the changes in the five CRLs that exhibited the largest increase in volume after disclosure. All five CRLs were inconsequential prior to the disclosure, and are composed of large commercial CAs. The commercial CAs, who service several popular websites and pay more servicing the CRLs, have a larger incentive to avoid the increased costs.

3.5 Notification

Three weeks after the initial disclosure a large number of hosts remained vulnerable, and we began notifying the system operators responsible for unpatched systems. This endeavor provided us with an opportunity to study the impact of large-scale vulnerability notification. In this section we describe our notification methodology and analyze the reactions to our notifications and its impact on patching.

3.5.1 Methodology

In order to find the appropriate operators to contact, we performed WHOIS lookups for the IP address of each vulnerable host appearing in our April 24, 2014 scan of the full IPv4 address space. We used the "abuse" e-mail contact extracted from each WHOIS record as our point of notification. We chose to use WHOIS abuse emails because they struck us as more reliable than emails from other sources. There also appeared to be less risk in offending a network operator through contacting the abuse contact. For example, many emails extracted from certificate Subject fields were not valid emails, and we observed several WHOIS records with comments instructing anything related to spam or abuse be sent to the abuse contact rather than the technical contact.

Our scan found 588,686 vulnerable hosts. However, we excluded embedded devices—which accounted for 56% of vulnerable hosts—because administrators likely had no avenue for patching many of these devices at the time. The remaining 212,805 hosts corresponded to 4,648 distinct abuse contacts. Approximately 30,000 hosts belonged to RIPE and Amazon each. Because neither of these organizations directly manage hosts, we omitted them from our notifications.

To measure the impact of our notifications, we randomly split the abuse contacts into two groups, which we notified in stages. We sent notifications to the first half (Group A) on April 28, 2014, and the second half (Group B) on May 7, 2014. Our notification e-mail introduced our research and provided a list of vulnerable hosts, information on the vulnerability, and a

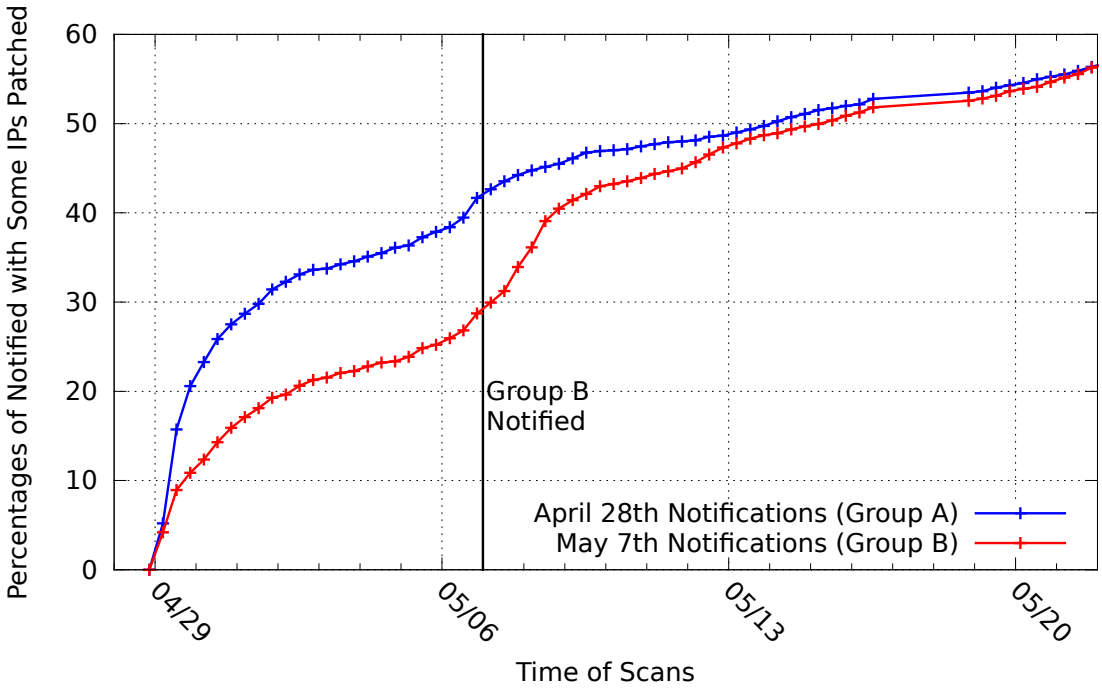

Figure 3.9: **Patch Rates of Group A vs Group B.** The patch rates for our two notification sets show that notification had statistically significant impact on patch rate.

link to the EFF's Heartbleed recovery guide for systems administrators.

3.5.2 Patching Behavior

To track patching behavior, we conducted regular scans of the known vulnerable hosts every eight hours. We considered a contact as having reacted and begun patching if we found at least one host in the list we sent to the contact as patched. Figure 3.9 shows a comparison of the patch rates between the two groups. Within 24 hours of the initial notification, 20.6% of the Group A operators had begun to patch, whereas only 10.8% of Group B contacts (not yet notified) had reacted. After eight days (just before the second group of notifications), 39.5% of Group A contacts had patched versus 26.8% in Group B. This is a 47% increase in patching for notified operators.

Fisher's Exact Test yields a one-sided p-value of very nearly zero for the null hypothesis that both groups reflect identical population characteristics. We thus conclude that our

notification efforts had a statistically significant positive effect in spurring notified sites to patch. Our second round of notifications followed a similar pattern as the first. As Group A's rate of patching had decreased at that point, Group B caught up, resulting in both converging to around 57% of contacts having reacted within a couple of weeks of notification.

We also investigated the relationship between the reactions of network operators (per Section 3.5.3) and their patching behavior. First, we sent our notification message in English, possibly creating a language barrier between us and the contact. We analyzed the Group A responses and found that email responses entirely in English had no statistically significant difference in the corresponding patching rate than for responses containing non-English text (Fisher's Exact Test yielded a two-sided p-value of 0.407).

We did, however, find statistically significant differences between each of the categories of responses framed below in Section 3.5.3, as shown in Figure 3.10, with human responders patching at the highest rate. Within the first day post-notification, 48% of human responders had begun patching, while none of the other categories had a patch rate higher than 32%.

The second strongest reactions came from contacts configured to send automated responses. 32% had reacted after one day, and 75% had reacted after three weeks. This indicates that operators using a system to automatically process notifications and complaints will still often react appropriately.

Over 77% of the contacts never responded. After one day, 20% of such contacts had conducted some patching; after three weeks, 59% had. Right before Group B's notifications, the patch rate of these contacts was statistically significantly higher than Group B's patch rate. This shows that even when system operators do not respond when notified, they often still patch vulnerable systems.

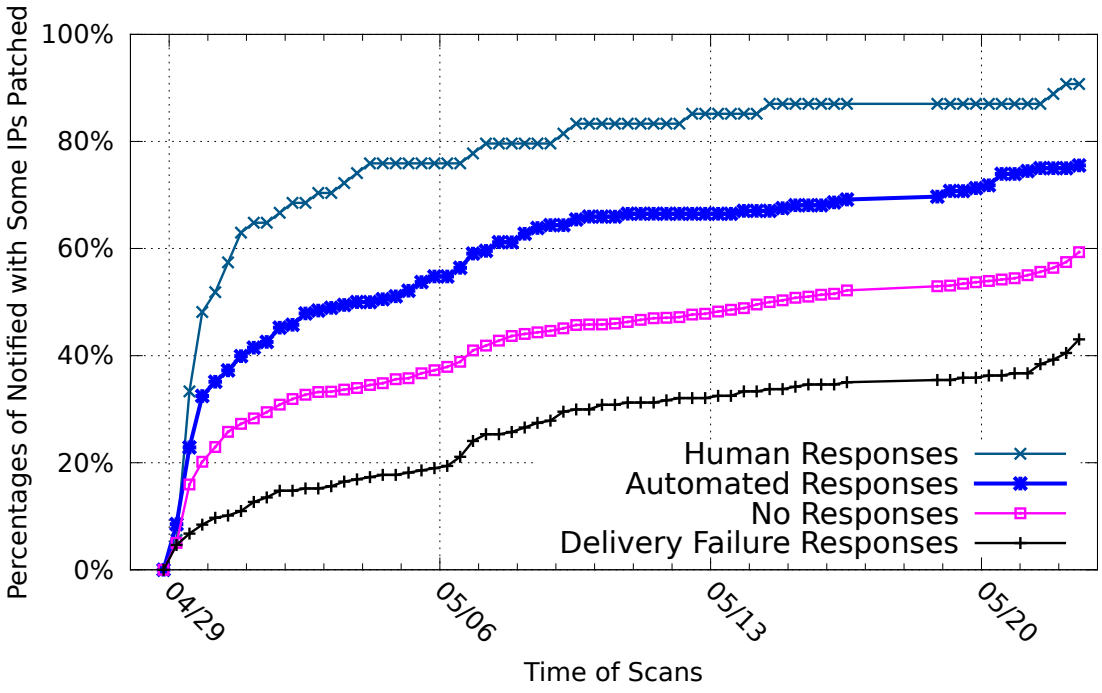

Figure 3.10: **Patch Rates for Different Response Types.** Conditioning on the sort of reply we received for a given notification reveals statistically significant differences.

3.5.3 Responses

In our first group of notifications, on April 28, 2014, we contacted 2,308 abuse contacts and received email responses from 514 contacts. Of these 59 (11%) were clearly human-generated, 197 (38%) were automated, and 258 (50%) were delivery failures. We received 16 automated emails where we subsequently received a human response; these we classified as human (thus, in total we received 530 emails). The vast majority of responses (88%) were in English; other common languages included Russian, German, Portuguese, and Spanish.

We classified a positive response as one that thanked us or stated their plan to remedy their vulnerable hosts. The human-generated responses were overall very positive (54/59), with only three that we deemed neutral, and two negative. The two negative responses questioned the legality of our scan despite our explicit explanation that we did not exploit the vulnerability.

Automated messages came in four forms: confirmations (24%), tickets (44%), trackers

(23%), and miscellaneous bounces (9%; primarily out-of-office notices and "no longer working here" messages). Confirmation emails confirmed the receipt of our notification; tickets provided a reference or ticket identifier to associate with our notification message; and trackers were tickets that also explicitly provided a link to a support site to track progress on opened tickets. Curiously, 21 of the 45 trackers did not provide the credentials to log into the support website, 2 provided invalid credentials, and 3 did not have our ticket listed on their support site. In the week following our notification, we were informed that 19 tickets were closed, although only 4 provided any reasoning.

Out of the 258 delivery failure replies, 197 indicated the recipient did not receive our notification. Other error messages included full inboxes or filtering due to spam, and several did not describe a clear error. We observed 30 delayed and retrying emails, but all timed-out within three days.

3.5.4 Network Operator Survey

We sent a brief survey to positive human responders, where all questions were optional, and received anonymous submissions from 17 contacts. Surprisingly, all 17 expressed awareness of the vulnerability and stated their organizations had performed some remediation effort prior to our notification, typically through informing their clients/customers and patching machines if accessible. When we asked why might the hosts we detected still be vulnerable, the most common responses were that they did not have direct control over those servers, or their own scans must have missed those hosts. It appears ignorance of the vulnerability and its threat did not play a factor in slow patching, although our sample size is small. When asked if they replaced or revoked vulnerable certificates, nine said yes, two said no, and one was unaware of the recommendation. Finally, we asked if these contacts would like to receive notifications of similar vulnerabilities in the future. Twelve said yes, two said no, and the others did not respond. This again demonstrates that our notifications were in general well-received.

3.6 Discussion

Heartbleed's severe risks, widespread impact, and costly global cleanup qualify it as a security disaster. However, by analyzing such events and learning from them, the community can be better prepared to address major security failures in the future. In this section, we use our results to highlight weaknesses in the security ecosystem, suggest improved techniques for recovery, and identify important areas for future research.

HTTPS Administration. Heartbleed revealed important shortcomings in the public key infrastructure that underlies HTTPS. One set of problems concerns certificate replacement and revocation. As discussed in Section 3.4, only 10% of known vulnerable sites replaced their certificates, and an astounding 14% of those reused the existing, potentially leaked, private key. This data suggests that many server administrators have only a superficial understanding of how the HTTPS PKI operates or failed to understand the consequences of the Heartbleed information leak. This underscores the importance for the security community of providing specific, clear, and actionable advice for network operators if similar vulnerabilities occur in the future. Certificate management remains difficult for operators, highlighting the pressing need for solutions that enable server operators to correctly deploy HTTPS and its associated infrastructure.

One of the ironies of Heartbleed was that using HTTPS, a protocol intended to provide security and privacy, introduced vulnerabilities that were in some cases more dangerous than those of unencrypted HTTP. However, we emphasize that HTTPS is ultimately the more secure protocol for a wide variety of threat models. Unfortunately, only 45% of the Top 1 Million websites support HTTPS, despite efforts by organizations such as the EFF and Google to push for global HTTPS deployment.

Revocation and Scalability. Even though only a small fraction of vulnerable sites revoked their certificates, Heartbleed placed an unprecedented strain on certificate authorities and revocation infrastructure. In the three months following public disclosure, about as many

revocations were processed by CAs as in the three years proceeding the incident. Wholesale revocation such as required by an event like Heartbleed stresses the scalability of basing revocation on the distribution of large lists of revoked certificates. As a result, CAs were backlogged with revocation processing and saddled with unexpected financial costs for CRL distribution—CloudFlare alone paid $400,000 per month in bandwidth [110]. The community needs to develop methods for scalable revocation that can gracefully accommodate mass revocation events, as seen in the aftermath of Heartbleed.

Support for Critical Projects. While not a focus of our research, many in the community have argued that this event dramatically underscores shortcomings in how our community develops, deploys, and supports security software. Given the unending nature of software bugs, the Heartbleed vulnerability raises the question of why the Heartbeat extension was enabled for popular websites. The extension is intended for use in DTLS, an extension unneeded for these sites. Ultimately, the inclusion and default configuration of this largely unnecessary extension precipitated the Heartbleed catastrophe. It also appears likely that a code review would have uncovered the vulnerability. Despite the fact that OpenSSL is critical to the secure operation of the majority of websites, it receives negligible support [89]. Our community needs to determine effective support models for these core open-source projects.

Notification and Patching. Perhaps the most interesting lesson from our study of Heartbleed is the surprising impact that direct notification of network operators can have on patching. Even with global publicity and automatic update mechanisms, Heartbleed patching plateaued two weeks after disclosure with 2.4% of HTTPS hosts remaining vulnerable, suggesting that widespread awareness of the problem is not enough to ensure patching. However, as discussed in Section 3.5, when we notified network operators of the unpatched systems in their address space, the rate of patching increased by 47%. Many operators reported that they had intended to patch, but that they had missed the systems we detected.

Although Internet-wide measurement techniques have enabled the mass detection of

vulnerable systems, many researchers (including us) had assumed that performing mass vulnerability notifications for an incident like Heartbleed would be either too difficult or ineffective. Our findings challenge this view. Future work is needed to understand what factors influence the effectiveness of mass notifications and determine how best to perform them. For instance, was Heartbleed's infamy a precondition for the high response rate we observed? Can we develop systems that combine horizontal scanning with automatically generated notifications to quickly respond to future events? Can we standardize machine-readable notification formats that can allow firewalls and intrusion detection systems to act on them automatically? What role should coordinating bodies such as CERT play in this process? With additional work along these lines, automatic, measurement-driven mass notifications may someday be an important tool in the defensive security arsenal.

3.7 Conclusion

In this work we analyzed numerous aspects of the recent OpenSSL Heartbleed vulnerability, including (1) patching behavior, and (3) impact on the certificate authority ecosystem. We found that the vulnerability was widespread, and estimated that between 24–55% of HTTPS-enabled servers in the Alexa Top 1 Million were initially vulnerable, including 44 of the Alexa Top 100. Sites patched heavily in the first two weeks after disclosure, but patching subsequently plateaued, and 3% of the HTTPS Alexa Top 1 Million sites remained vulnerable after two months. We further observed that only 10% of vulnerable sites replaced their certificates compared to 73% that patched, and 14% of sites doing so used the same private key, providing no protection.

We also conducted a mass notification of vulnerable hosts, finding a significant positive impact on the patching of hosts to which we sent notifications, indicating that this type of notification helps reduce global vulnerability. Finally, we drew upon our analyses to frame what went well and what went poorly in our community's response, providing perspectives on how we might respond more effectively to such events in the future.

CHAPTER IV

CAge: Taming Certificate Authorities by Inferring Restricted Scopes [74]

4.1 Introduction

Every day, millions of Internet users rely on HTTPS to secure transactions such as online banking, e-mail, and e-commerce against malicious eavesdroppers or tampering through man-in-the-middle attacks. HTTPS combines the Transport Layer Security (TLS) protocol with a public-key infrastructure (PKI) based on certificate authorities (CAs) that are trusted by the browser. Each server presents its public key in the form of an X.509 certificate corresponding to its domain name and digitally signed by a CA, which is responsible for verifying the identity of the website, usually for a small fee. Browsers maintain a set of trusted root CAs and subsequently trust the purported identities of certificates signed by any CA in this trusted set. In addition, these root CAs are typically able to sign certificates for additional CAs, known as intermediate certificate authorities, which are trusted recursively by the browser.

CA-signed certificates help protect users in the presence of man-in-the-middle adversaries, but they cannot protect users from compromise of the CAs themselves. In recent years, there have been several high-profile attacks on CAs that resulted in the signing of fraudulent certificates. For instance, in 2011, an attacker breached the security of a rela-

tively small Dutch CA named DigiNotar and created certificates for dozens of popular sites, including `*.google.com` [29]. An ISP in Iran subsequently abused this latter certificate to conduct man-in-the-middle attacks against Google services. Of course, attackers do not require a malicious or compromised ISP to utilize illegitimate certificates; they can also use DNS cache poisoning attacks [73], intercept and modify traffic on an insecure wireless network, or use ARP spoofing on a local subnet.

Preventing DigiNotar-style attacks is difficult, because there are currently very few technical restrictions on what trusted CAs can sign for—once they convince a browser they are trustworthy, they are given an unrestricted capability to vouch for any domain name they choose. This ability leads to an enormous attack surface: an attacker who compromises any one of over 1200 CAs can then impersonate any website that relies on HTTPS. The status quo violates the principle of least privilege: DigiNotar should not have had the capability to sign certificates for Google, nor should a CA run by a small university in the United States be allowed to sign certificates for another country's intelligence agency, such as a website in the `.gov.ir` domain. In other words, each CA's trust should come with a limited scope.

One way to limit the scope of CA trust is to designate a set of top-level domains (TLDs), such as `.com` or `.uk`, within which each CA may sign. Indeed, we present data that suggests that most CAs currently only sign certificates for sites in a small number of TLDs, and conversely, that sites in most TLDs utilize only a small set of CAs. Many CAs appear to sign exclusively for domains belonging to a single organization, and others appear to operate within a specific country, sector, or both. Although this suggests that TLD-based restrictions could be fruitful, realizing them within the existing PKI is a challenge. The X.509 name constraints extension (see Section 4.3) introduced the ability to explicitly declare such restrictions in new CA certificates, but it requires participation from each root or intermediate CA, as well as implementation in all client systems, and has seen almost no adoption.

Rather than relying on each CA to explicitly declare a TLD scope, we explore the

possibility that browser makers could *infer* such scopes without CA participation. We propose a mechanism called CAge that creates a profile of each CA based on the TLDs of publicly visible certificates it has previously signed. If a browser later encounters a certificate for a site in a TLD a CA has never been observed to sign before, this can be treated as evidence of suspicious behavior. These restrictions can be implemented without cooperation from the certificate authorities, at the risk that CAs will change their behavior over time and begin signing for certificates outside their previous pattern. Empirically, however, we find that this rate of change is quite low, that inferred scopes generated with simple algorithmic rules would result in a low false-positive rate, and that the CAge approach would allow browser makers to dramatically reduce the attack surface of the HTTPS PKI.

Outline We begin in Section 4.3 with a discussion of related work. In Section 4.4, we analyze data from an Internet-wide survey of HTTPS certificates to examine the current practices and distribution of CA's signing. Supported by evidence from this dataset, in Section 4.5, we give a detailed description of the CAge approach for inferring TLD-based restrictions. In Section 4.6, we quantitatively evaluate CAge's performance, and in Section 4.7 we describe a prototype implementation in the form of a browser extension. Finally, we conclude in Section 4.8.

4.2 Background

In this section, we present a brief background of X.509 and its extensions that are relevant for this paper. For a more in-depth background on the TLS public key infrastructure, we recommend RFC 5280 [36].

When a browser connects over HTTPS, the server presents an X.509 certificate. The certificate includes an identity (such as the domain name of the server), a validity period, a public key, and a digital signature over the rest of the certificate. The browser checks that the certificate's identity matches the domain the browser is attempting to connect to, that the

certificate is still valid, and that the digital signature of the certificate is correct. The public key is then used to authenticate or transmit a shared secret with the server, which is, in turn, used to establish an end-to-end secure channel.

The certificate's digital signature is signed by a trusted Certificate Authority (CA) which is in charge of verifying the identity of the website. When the server presents its certificate, it also includes the certificate of the CA responsible for signing. The signing-CA's certificate may also in turn have been signed by another CA, in which case, this second CA is also included in the bundle of certificates sent to the browser. This bundle of certificates is referred to as a *certificate chain*, and the browser only trusts it if one of the CAs in the chain is in a trusted set of root CAs that are built into the browser.

4.3 Related Work

Given the documented problems with CAs [29, 112, 134, 135], it is not surprising that providing authentication on the Internet is an active research topic. Many researchers have suggested using new authentication techniques or making changes to the authentication infrastructure. Multi-path probing [21, 88, 129] has been suggested as a way to move away from certificate authorities. The technique necessitates the availability and access to trusted notaries, which presents problems for captive portals and sites presenting multiple certificates. The suggested changes to the CA infrastructure focus on making certificate signing transparent and publicly verifiable [46, 82]. The work needs widespread support with implementation and testing before the effects of these systems can be seen. CAge instead works with existing authentication mechanisms and does not require collaboration with CAs or additional infrastructure.

Quick local improvements on the security of the current CA system have also been proposed and adopted in limited form. The idea of certificate pinning, where the browser remembers which certificates belong to each domain, has been used to various degrees by existing software. Google Chrome uses certificate pinning for Google's own websites [52],

while CertPatrol [85], a Firefox extension, allows the user to pin all certificates that they encounter and warns the user when a certificate changes. However, CertPatrol's fine granularity of control, frequent false positives, and required knowledge deters all but the most dedicated users. Soghoian and Stamm proposed CertLock [119], which pins the country code of the CA to the domain. The design is aimed at preventing compelled certificate creation attacks in which a government forces a CA to issue false certificates. The technique can work fairly well for U.S. domains being attacked by foreign governments; however, due to the dominance of the U.S. in the CA market, the proposal's true effectiveness is uncertain. Our solution, CAge, pins information about the CA rather than the specific domain and does not demand additional knowledge from the user. CAge does not protect against a specific attack, but rather aims to decrease the attack surface of the PKI in general.

Scopes on certificate authorities have also been proposed through X.509 Name Constraints. X.509 Name Constraints [36] is a certificate extension with the ability to restrict CAs to a particular set of domains. All trusted certificates must conform to the name constraint extensions in their certificate chain. Implementation and adoption of the accepted standard has been slow. There is only one trusted CA that is currently constrained with the extension, the "Hellenic Academic and Research Institutions RootCA 2011", which was adopted in Firefox 11.0 [9]. Name constraints' reluctant adoption is likely due to the high cost of enforcement. Since browsers only have direct control over their root CAs, browsers would have to force tight constraints directly on the root CAs. Complicating matters, root CA certificates have an average lifetime of more than 20 years. Long certificate lifetimes require these constraints to be forecast far into the future, which is both difficult and prone to error. Wide-scale adoption of name constraints also requires all existing certificates to be reissued under constrained CAs, requiring a long-term transitional plan. CAge is able to leverage knowledge of the complete CA system and is able to differentiate the constraints between roots and specialty intermediaries. Root CAs are tightly constrained to their intended purpose and intermediaries are known and constrained to their previously demonstrated

behavior. CAge can be applied immediately by browsers without CA collaboration, making immediate adoption feasible.

4.4 Analyzing the CA Infrastructure

Currently, important aspects of browser trust behavior and CA signing practices are surprisingly opaque to outside observers. For example, certificate chaining allows browser-trusted root CAs to delegate their signing authority to third parties by issuing intermediate CA certificates; this can be done in private and at any time, making it impossible to determine the full trusted set of CAs by analyzing browser software alone. Furthermore, CAs typically do not publish the set of domains for which they have issued certificates, making it difficult to study their patterns of signing behavior in practice.

To understand these aspects of the HTTPS PKI, we analyzed a large corpus of certificates collected with an Internet-wide scan of HTTPS servers recently conducted in a study by Heninger et al. [63] The study exhaustively probed the IPv4 address space on TCP port 443 (the default port for HTTPS) and collected every certificate chain presented by a responding server. The resulting data, from October 2011, provides a recent and comprehensive view of the TLS certificate landscape. It includes responses from 7.7 million hosts, which returned more than 5.8 million distinct certificates.

First, we determined which of the certificates would be trusted by the major web browsers and extracted the set of intermediate CAs that issued them from the provided certificate chains. We started with the 317 root and intermediate CAs[1] that are directly trusted by Mozilla, Apple, Microsoft, and OpenSSL (Google Chrome defaults to the platform's trusted key store). We then used the OpenSSL API to test each of the collected certificate chains for validity. We deemed a certificate valid if it was not expired at the time of the scan and there existed a chain of valid CA certificates rooted by a directly trusted CA certificate. In all, there were 1,956,267 valid certificates (comprising 2,558,492 unique domain names) issued

[1] Unless otherwise noted, we distinguish CAs by subject and subject key identifier.

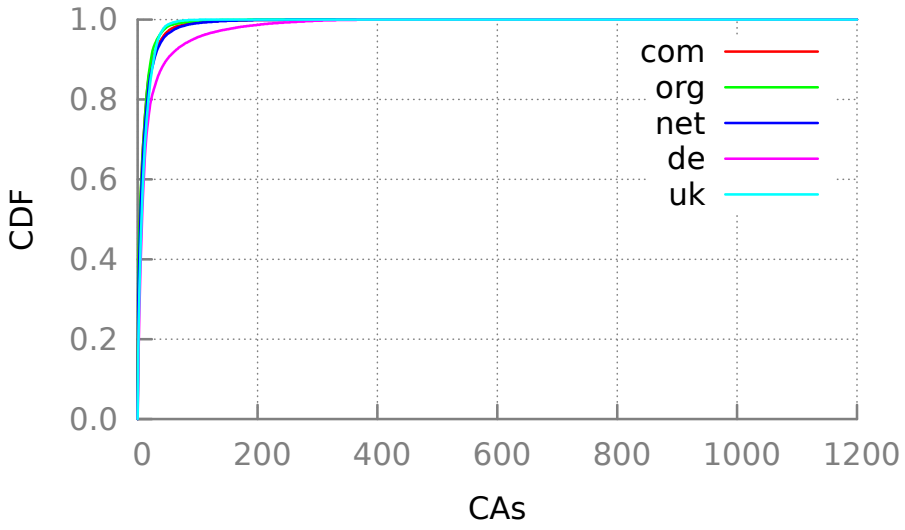

Figure 4.1: **CAs Signing for Top 5 TLDs** — This figure shows the cumulative distribution of signed domains within the top five TLDs and how many trusted CAs accounted for each fraction. A small number of large CAs dominate.

by 1207 CAs.

The number of certificates signed by each CA varied considerably. The top 20 CAs were responsible for more than 80% of valid certificates. Figure 4.1 shows the cumulative distribution of signed domains within the top five TLDs and the number of CAs responsible for signing each fraction. Over 90% of all signed `.com` domain names used certificates issued by just 25 certificate authorities.

Despite this lop-sided distribution of CA size, each of the 1207 CAs had the ability to issue trusted certificates for any domain name. To examine how much of this authority each CA exercised, we extracted the set of domain names that each CA had directly issued certificates for, and then examined the set of TLDs to which these domains belonged. We found that 89% of CAs had signed for domains in fewer than 10 unique valid TLDs [66], with the majority (65.8%) of CAs signing for domains in either zero or one TLD.[2]

Figure 4.2 shows the top 25 TLDs by number of signed domains, together with the fraction of total signed domains that are in each TLD and the number of CAs that issued

[2] A CA can sign for zero valid TLDs if it does not sign domain certificates directly but instead signs other intermediate CA certificates, for example.

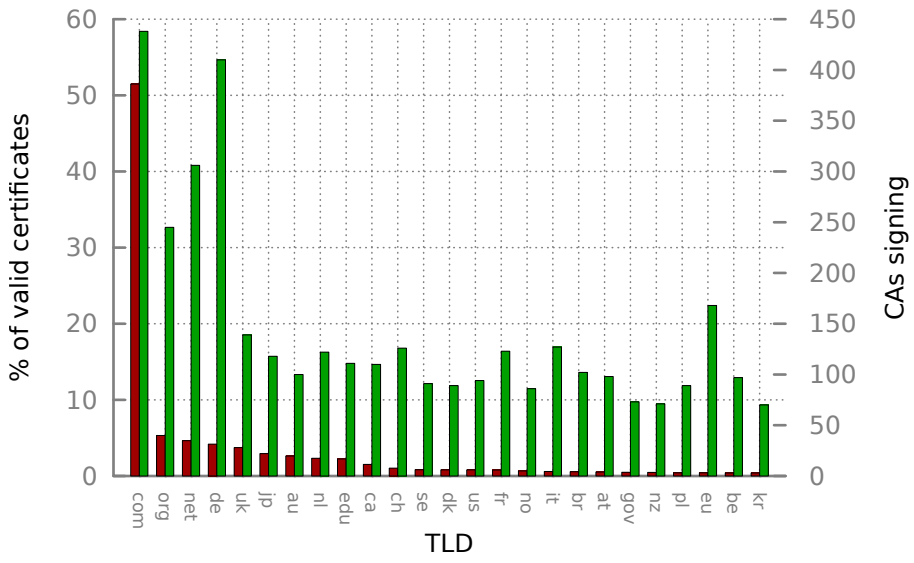

Figure 4.2: **Top 25 TLDs for Signed Domains** — This graph shows both the fraction of signed domains falling within each of the top 25 TLDs (*red*) and the number of CAs that sign for at least one of these domains (*green*). Although `.com` is by far the most common TLD, 65% of CAs have never signed a certificate for a `.com` domain. Less common TLDs include signed domains from even smaller fractions of CAs, suggesting that the ability to sign for *all* domains is unused by the vast majority of CAs.

certificates within each TLD. Although `.com` accounts for 51% of signed domains, fewer than 35% of trusted CAs had signed a certificate for even a single `.com` domain, and only 20% had signed for 10 or more such certificates. There were 787 CAs that had *never* signed for a `.com` domain. Similarly, fewer than 11% of CAs had signed certificates in the `.uk` TLD, and only 6.6% had signed for 10 or more in the `.uk` domain.

To better understand why most CAs seem to be issuing certificates within so few TLDs, we manually investigated many of the trusted CAs. One reason for these sparse signing practices is that many of the CAs belong to private companies and organizations and are used for domains under their control. For instance, more than 200 German universities and research institutions are browser-trusted CAs; their impact on statistics for the `.de` TLD is clearly visible in Figures 4.1 and 4.2. Many corporations including Ford, Disney, Wells Fargo, and Migros also have trusted CA certificates. We observed that they generally limit

their signing practices to a few specific second-level domains. For instance, the Walt Disney CA signs almost exclusively for domains under `*.disney.com`. However, we note that some of these CAs may sign for additional private domains that do not show up in our data set of publicly accessible HTTPS servers.

Another reason appears to be that many smaller CAs focus their business within a specific geographic region and tend to sign domains under a country-specific TLD. (We noted that 29% of CAs signed for domains within only a single TLD and not in `.com`.) For example, the public AusCERT CA signs 97% of their certificates under the `.au` and `.nz` TLDs, and the Coop Swiss company signs exclusively within `.ch`. The infamously compromised CA, DigiNotar, specialized in the Dutch market and issued 93% of its certificates to `.nl` domains.[3]

4.5 Our Proposal

Our analysis in the previous section suggests that the vast majority of CAs do not need or use the authority they have—to issue a trusted certificate for any domain in any TLD. Leaving these CAs with such unconstrained signing power leaves Internet security unnecessarily vulnerable: an adversary can choose the weakest of over 1200 CAs to attack in order to gain complete signing authority for any domain. In this section, we propose CAge, a browser-based approach that restricts CA signing to TLDs in which they have *already* signed. We argue that this would improve the ecosystem security of the HTTPS PKI without impairing how it is used today.

CAge consists of two phases, an initialization phase and an enforcement phase. In the initialization phase, we collect certificates from an Internet-wide scan and infer rules from the observed current CA signing practices. Browsers then deploy CAge as a browser extension in the enforcement phase to restrict CAs to these inferred scopes and handle

[3]DigiNotar was already defunct at the time of our scan, so this figure is based on December 2010 data from the EFF SSL Observatory [49].

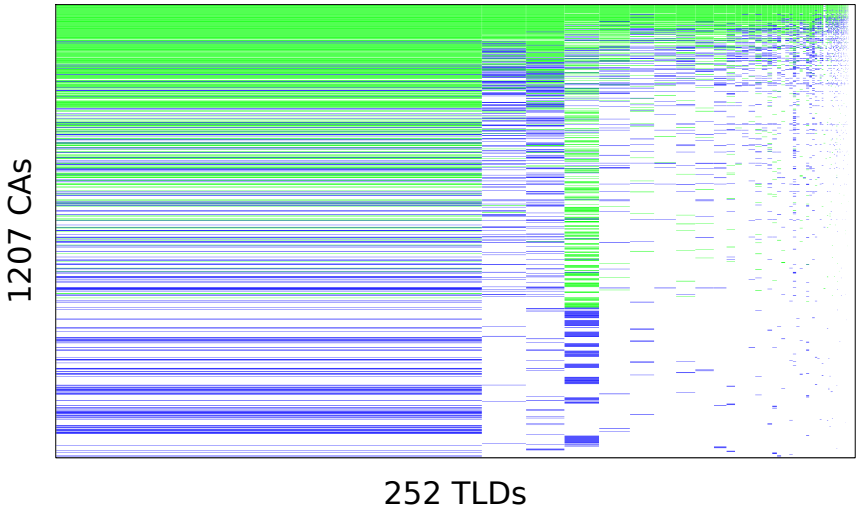

Figure 4.3: **CA/TLD Matrix** — This figure shows a matrix of CAs that have signed for certificates in TLDs. Each position is colored if the corresponding CA (row) has signed for at least one (blue) or ten (green) domains in the corresponding TLD (column). Each axis is sorted by total domains signed, putting the most prolific CA at the top and most common TLD at the left. The width of the TLD columns are scaled by the percentage of certificates made up in that column—.com is visible as the left-most column, as it accounts for over 50% of the total domains signed. A significant portion of this matrix is empty, illustrating the sparse nature of CA signing practices.

exceptions. We describe these phases in more detail below.

4.5.1 Initialization and Rule Inference

Prior to deploying CAge, the browser maker needs to develop an initial set of restricted scopes to apply to existing CAs. However, creating justifiable rules for existing CAs necessitates knowledge of the current CA market. Given the distributed design of the CA system, a comprehensive scan of HTTPS must be performed like the one completed by Heninger et al. [63]. Such a scan can be used to determine the observable list of intermediate certificate authorities, as well as the domains for which they have directly signed certificates.

After scanning and collecting the raw data, we infer rules and restrictions for the CAs, based on current practices. As stated earlier, there are many CAs that have never signed for particular top-level-domains. If the user was presented such a certificate, there is a

high probability that the certificate is fraudulent and the user should be alerted. As a first approach, CAge can generate the inferred scopes by looking at the TLDs for which each CA has previously signed. If a CA has signed for any domains in a particular TLD, the inferred rules will allow the CA to continue to sign for that TLD, which will be referred to as the TLD policy.

Rules are stored for each CA as a set of regular expressions that governs the domains they are allowed to sign. For example, a CA may have the regular expressions `.*\.au` and `.*\.nz`, allowing it to sign for two TLDs. This allows for the rule inference to be extendable to other algorithms in the future. In general, rules should be generated from an algorithm taking the CAs and their signed domains as input and producing the CA restrictions as output. CAs could be constrained to second-level domains or more specific rules could be required for larger TLDs, factoring in the cost of false positives, and both the size and brittleness of the rule set. We explain variations on our generation algorithm in Section 4.6.1.

4.5.2 Enforcement and Exception Handling

Once CAge has inferred CA signing rules from the collected scans, CAge relies on browsers to enforce these rules during certificate validation. Browsers have a strong incentive to protect their users from fraudulent certificates, making them a natural place to enforce these restrictions.

Normally, browsers verify that HTTPS certificates have a valid signature chain to a trusted root. With CAge, browsers additionally compare the domain to the set of regular expression rules inferred for that certificate's intermediate (signing) CA. If the domain does not fall within the allowed rules for the given CA, CAge alerts the user with a warning, explaining that the website's origin is certified by an unusual source. CAge also asks the user if they want to send the violation to the browser maker for further inspection. This feedback allows the browser to potentially verify the authenticity of the certificate via other means, while respecting the user's privacy. The browser may use techniques such as multi-path

probing [129] to aid the user in authenticating the domain.

4.5.3 Updating

Keeping the rule set accurate and current is crucial to keeping a low false positive rate and avoiding user habituation to clicking through warning messages. The CAge rule set must be updated as CA policies change and new CAs emerge. Luckily, browser makers are in a good position to provide updates to users, based on newly discovered certificates reported collectively by users. Updates to the CA signing rules can be pushed to users through standard browser update mechanisms.

Any update mechanism based on inferred rules runs the risk of being gamed by attackers; for example, if an attacker compromises a CA that is not currently allowed to sign for a domain in the `.com` TLD, the attacker can request that the CA sign a legitimate `.com` domain, owned by the attacker. Under a naive rule set update approach, inferred TLD rules will soon change to allow this CA to sign for the `.com` TLD, and the attacker can use the previously compromised CA to sign for other `.com` domains fraudulently. This means the browser maker cannot simply look to long-standing valid certificates in order to validate new TLD rules once an attacker knows of the system. Although the browser maker could query certificate authorities about the legitimacy of signings and changes to rules, attackers can still increase the scope of all publicly-signing intermediate certificate authorities. A CA might not turn away business for a domain simply because they have not signed for the top-level domain in the past. While CAge would still protect users from illegitimate certificates signed by certificate authorities that do not sign publicly (including private organizations, root CAs and inactive intermediates), inferred TLD updates would severely increase the attack surface.

For this reason, the CAge rule set is updated on a per domain basis. When a domain exception is reported, the domain is added to a "watchlist" and is manually verified before the specific certificate's domain is whitelisted and pushed as an update. We show in Section 4.6.2

that these updates are infrequent and thus allow manual inspection and verification.

New CAs, without any recorded behavior, may specify their intended scope to the browser maker in advance to avoid the whitelist update mechanism for their domains. While this might be seen as an additional hurdle for new CAs entering the market, ultimately, the authority to say whether or not a particular CA is trusted lies with the browser.

4.6 Evaluation

In this section, we quantitatively evaluate CAge's performance in experiments based on real world data.

4.6.1 Attack Surface Reduction

CAge reduces the overall attack surface by restricting the scope of certificate authorities, but a metric is required to evaluate the effectiveness of CAge quantitatively and to compare various rule inference algorithms. Our goal is to quantify the relative risk of damage that could be caused by an attacker-compromised CA. Treating each signed valid domain as a protected entity, we approximate the attack surface by:

$$\sum_{c \in CAs} domains[c]$$

where $domains[c]$ is the number of domains that a given CA c can sign. Currently, all CAs can sign for all domains; therefore, $domains[c]$ is equal to the number of signed valid domains for all c.[4] Under the CAge policy that restricts CA scopes by TLD, domains[c] is the total number of valid domains across all the TLDs that c is permitted to sign. For example, if a CA is allowed to sign for only .com because they signed for 100 of the 1.3 million .com domains in the dataset, $domains[c]$ for that CA would be 1.3 million.

While this attack surface metric is by no means complete, it does provide a simple and

[4]There were not any trusted CAs that contained name constraints in the November 2011 scan.

intuitive first-order approximation that allows us to compare the relative risks of different CA restriction policies quantitatively.

As mentioned in Section 4.5.1, a very simple approach would be to only allow a CA to sign for top-level domain names the CA had previously signed, the TLD policy. Applied to our data set, the TLD policy results in a 75% decrease in attack surface than current practice.

We can improve this result by modifying the inference procedure to only allow a CA to sign for domains in a TLD if it has previously signed for a minimum threshold t of unique domains in that TLD. Domains signed by CAs that do not meet the policy cut-off can either be viewed as suspicious anomalies or manually inspected, labeled as an exception, and whitelisted within the database. Figure 4.3 provides a visualization of this attack surface for two policies. When requiring the CA to sign for 25 domains before allowing authorization over the complete TLD ($t = 25$), the attack surface is reduced to 11.1% of the original. Figure 4.4 shows the attack surface of the TLD policy as it becomes more strict.

CAge may also be evaluated based on its ability to stop recent CA compromises. In the Comodo attack that occurred in March 2011 [112], an attacker issued fraudulent certificates for `.com` domains signed by "CN=UTN-USERFirst-Hardware", a relatively large CA which had signed over 25,000 other `.com` certificates previously. Due to these signing practices, CAge would have been unable to protect against the Comodo attack. Similarly, all but two of the top 20 CA certificates have signed domains from over 100 unique TLDs, limiting the usefulness of restricting these large CAs to the TLDs they currently sign.

However, the vast majority of CAs do not sign for such a diverse market allowing CAge to provide protection during a CA compromise. For instance, CAge would have detected the DigiNotar compromise. The EFF's SSL Observatory [49] data, which was collected a year before the attack, shows that the issuer of the fraudulent `*.google.com` certificate, "DigiNotar Public CA 2025" [8], had not signed certificates for any `.com` domains. Had CAge been implemented at the time, it would have prevented the attacker from using the `*.google.com` against Internet users in Iran.

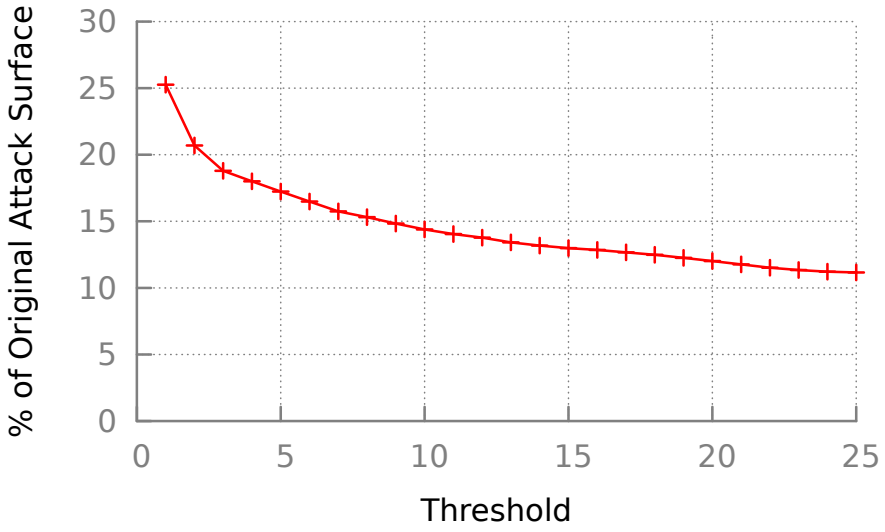

Figure 4.4: **Attack Surface Metric** — HTTPS PKI Attack Surface under the basic TLD rule inference policy. Threshold refers to the number of domains signed before the TLD is added to the CA's scope. Even restricting CAs to the TLDs they have previously signed reduces the attack surface to 25% of the status quo.

4.6.2 CAge Durability

Although the attack surface metric provides a quantifiable goal, it should not be the only factor when considering the inferred rule set. The minimum attack surface can be attained by pinning every domain to the CA that signed their certificate in the data set. This policy would provide the minimum attack surface, but the constraints would be very brittle and CAge would require a very large rule set that needed to be updated frequently. In addition, excessive false positives are not tolerable in any system, as users learn to ignore warnings [123]. Simply finding the policy with the minimum attack surface is not enough; CAge should attempt to capture the CA's de facto policies to avoid brittleness of the rule set.

In order to test the durability of our inferred rules, we acquired a second data set in April 2012 (6 months after the original scan) to test the viability of our solution. We found that the large majority of domains observed in newly issued certificates conformed to our rules, supporting our hypothesis that the TLDs that CAs sign for are generally stable. The basic TLD policy with $t = 1$ accommodated 99.84% of new certificates. Table 4.1 summarizes

TLD	Violating CAs	Violating Domains	Total Issued Domains	% Violated
com	10	27	519174	0.005
org	12	22	50531	0.044
net	14	28	46300	0.060
de	8	30	34160	0.088
uk	5	8	33113	0.024
jp	3	4	30699	0.013
au	3	3	21768	0.014
edu	2	3	20498	0.015
nl	6	20	19076	0.105
ca	5	6	16716	0.036
Total	**152**	**1506**	**937137**	**0.161**

Table 4.1: **TLD rule violations** — For the top 10 TLDs, we evaluated the certificates seen in April 2012 that contained domains violating inferred rules generated from data collected in October 2011 using a TLD policy with $t = 1$. Violating CAs represents the number of CAs that were not previously seen signing for this TLD in October 2011, but were observed signing for it in April 2012. Violating Domains represents the number of unique domains issued by Violating CAs in that TLD, while total issued domains represents the number of domains observed in new certificates seen in April. As shown by the percentage violation (domains / total issued domains), the vast majority of new certificates conform to the generated rules.

new certificates for domains in the top 10 TLDs. Our results show that only a handful of CAs signed for TLDs they had not previously signed. While several smaller and less stable TLDs had certificates with domains that violated our previous rules, all of the violations could be fixed by simply whitelisting the 1506 domains in the browser's rule set. CAge required an average of fewer than 10 domain updates per day over the 6 month period. Using a more restrictive TLD policy with $t = 25$ results in 10,035 violating domains (98.93% conformed).

4.7 Implementation

We have developed a CAge prototype in the form of a Firefox extension. We implemented the simple lenient policy, described in the previous section, in which certificate authorities can sign for any TLD they have previously signed. Better policies could be

obtained, but this simple policy is unlikely to produce many false positives, and achieves good theoretical results. The CAge extension downloads the current set of rules and known CAs from our CAge server over HTTPS upon installation. CA certificates are identified by their SHA1 fingerprint and rules are stored as regular expressions within the extension. When the browser is presented a TLS certificate, the browser forwards it to the CAge extension where it is matched against the current rules database. In addition to the standard rules table enabling the issuer to sign certificates, the CAge extension also contains tables for globally blacklisted CAs, locally ignored hosts, local rule sets, a local certificate white list, and a session certificate white list. These tables allow for the customization and protection against known threats to which users have grown accustomed.

If the extension cannot find an appropriate rule for the incoming certificate, it stops the connection and sends a request for updates to the CAge server over HTTPS[5]. The request contains a time stamp of the last extension update and synchronizes with the server if it is out of date. If the extension is still unable to resolve the certificate, it prompts the user, asking if they would like to query the CAge server with the domain name for more information.

The CAge prototype appears to be non-intrusive in our daily use. It has been used for normal browsing for four months and we have observed zero false positives while using the TLD policy with a threshold of one.

4.8 Conclusion

In this chapter, we presented CAge, a mechanism for inferring TLD-based restricted scopes for HTTPS CAs. Based on the empirical observation that the vast majority of browser-trusted CAs do not utilize their technically unconstrained signing power, we argue that each CA should be restricted to signing for domains within a limited set of TLDs. We show how such restrictions can be realized in practice by profiling past CA signing behavior, and we find that such an approach would dramatically reduce the attack surface of the

[5]The CAge extension uses a pinned certificate to avoid MITM attacks using fraudulent certificates

HTTPS PKI with a low rate of false alarms over time.

While browsers have a positive record of revoking compromised CA certificates once a breach is discovered, we believe much more can be done to proactively mitigate the damage caused by attacks against CAs and to provide defense-in-depth to the HTTPS PKI. Given the relative ease with which CAge could be deployed by browsers, we strongly encourage browser developers to adopt this approach to help combat the growing threats that HTTPS users face.

CHAPTER V

Let's Encrypt

5.1 Introduction

HTTP has seen tremendous success, but it is inherently insecure. HTTP is vulnerable to network attacks, including session hijacking, surveillance, and fine-grained censorship [57, 59,108]. HTTPS defends against network attackers by providing confidentiality, authenticity, and integrity to HTTP. Although there have been several vulnerabilities found within the protocol and implementations of TLS recently [5, 20, 40, 96, 115], by far the largest problem with HTTPS is it has not seen universal adoption. We found that only 12.9% of sites had enabled HTTPS with a trusted certificate in our scans of the Alexa Top 1 million [43]. HTTPS's relatively small deployment can be attributed to pure economics. The cost and benefits are borne upon two different parties. System administrators bear the cost of deploying TLS while the benefits are seen by users who are often unable or ill-equipped to modify their behaviour based on the security of the server. Further, like most security features, the value of TLS deployment is difficult to determine. The risk and cost of an attack is not easily estimated. We can attempt to fight economics and continue to try and force system administrators to internalize the benefits of HTTPS deployment, or we can yield to market forces and simply reduce the costs of deploying HTTPS. Through surveys of system administrators, we discovered two primary impediments to deploying HTTPS: the financial cost of the TLS certificate and the system administrator time necessary to setup

HTTPS. Ideally, system administrators should not feel there is a cost at all. In order to see wide-scale adoption of HTTPS and the long-term security of the Internet, we *must* reduce the cost of deployment to virtually zero. Configuring TLS on servers should be effortless for system administrators and financially free.

Reducing the monetary price of certificates depends on reducing the cost of issuing certificates. If we assume a perfectly competitive market, or a benevolent certificate authority, the certificate authority should charge the marginal cost of issuing the certificates. There are many fixed costs for certificate authorities. The CA must buy the required infrastructure and pay for the yearly audits, but the marginal cost of issuing each certificate is extremely small. Issuing 1,000,000 certificates is not much more expensive than issuing 100. The marginal cost of the CA only encompasses the validation of the domain names and the necessary increased bandwidth. If the validation mechanism can avoid human interaction, the marginal cost of issuing a certificate is practically zero. In Section 5.2.2 we evaluate the current validation mechanisms in use by CAs.

Reducing the system administrator time necessary to manage HTTPS is more complicated. There are several steps to setting up HTTPS, and setting up HTTPS varies across devices. First, system administrators must figure out that they need a certificate and navigate the convoluted market; snake-oil terminology is rampant, and prices of certificates range from free to several hundreds of dollars. The user will have to generate a private key, generate a certificate signing request (CSR), and perform the verification steps required by the CA. Then the user must figure out how to configure their server with the certificate and certificate chain file to deploy HTTPS. All of these steps involve esoteric commands and they often require the user to blindly follow a tutorial. This process takes system administrator's over an hour to attain and install a certificate on their system, a task which generally must be completed every year. If the system administrator forgets to renew their certificate, users will encounter certificate warnings when accessing the site, driving away business.

Assuming the user does manage to deploy HTTPS, they often do so incorrectly. Sites

are often misconfigured and do not support the latest protocol and cipher suites. We discovered that only 45% of server certificate chains were optimally configured and that 12.7% of sites serving once-valid browser-trusted certificates are misconfigured in a manner such that they are inaccessible to some modern clients [43]. We found that of the Alexa Top 1 Million domains that support HTTPS, 40.9% only support TLS 1.0 [40], and we found that only 44% of connections supported forward secrecy [96]. Supporting the latest protocols and cipher suites is important to maintain the security of HTTPS connections. Furthermore, vulnerabilities are regularly discovered within TLS that necessitate server modification. [5, 40, 96]. In our analysis of Heartbleed, we experimented with system administrator notifications and found them surprisingly effective. We observed a 47% increase in patching over the control group. System administrators are willing to patch, though they may need prompting either because they are unaware that they are vulnerable or because they may need detailed instructions. The servers themselves should be able to provide updates to the configurations based on newly publicized vulnerabilities in the same way that users can receive security updates from their package managers. Setting up and deploying HTTPS safely and correctly cannot be laborious.

In order to achieve our goals, humans must be removed from both the CA and client processes. Everything must be automated: attaining the certificate, the initial HTTPS setup, renewal, and the necessary configuration updates. Current certificate authorities generally charge for the service and we must rethink their process to create an equivalent service that is completely automated. In this Chapter, we present Let's Encrypt, the first completely automated and free certificate authority. Through the development of a new protocol, ACME, we have developed a method to allow users to deploy HTTPS with a single command, and we have reduced the time taken to deploy HTTPS from an hour down to 30 seconds.

Let's Encrypt is run by the Internet Security Research Group (ISRG), and it launches to the public the week of November 16, 2015.

In Section 5.2, we describe the current process for validating, attaining, and installing a

certificate for HTTPS. In section 5.3, we will describe ACME, a new protocol under IETF review that automates all aspects of certificate authorities. In Section 5.4, we describe and discuss the merits of the challenges. In Sections 5.5, we attempt to solve other long-standing HTTPS PKI issues. Finally, in Section 5.6, we discuss the current ACME implementations used by Let's Encrypt, and we describe the process of becoming a CA in Section 5.7.

5.2 Status Quo

Setting up TLS for a web server is surprisingly difficult. Users generally have to rely on a manual that is fraught with esoteric terms and instructions. In general, the first step is to find a trusted "certificate authority," which in and of itself can be confusing. Prices of certificates can range from free to thousands of dollars, and CA websites are notoriously plastered with spurious marketing. Assuming that the user has decided on a CA, the CA generally requires that the user first register and submit all of their information. The CA will then request a certificate signing request (CSR). The user must figure out that they need to generate a public/private key pair and the associated CSR, generally through command line tools.

Next, the certificate authority needs to prove that the user has authority over the domain. Typically, this is done by sending a special token via email to the WHOIS contact. The user will respond to the email and then wait a few minutes to a few hours for the final authorization. Once authorization is complete, most certificate authorities will require a form of payment before sending instructions for downloading the certificate and its associated certificate chain.

Once the user has downloaded the necessary files, they will again follow another guide for setting up TLS on their specific web server. This requires enabling the necessary modules for HTTPS, making the web server listen on port 443, installing the certificates appropriately, and configuring TLS. The system administrator must determine which protocols should be allowed and which cipher suites should be supported. The guides often fail to provide

instructions for desirable optional features like HSTS, OCSP stapling, and performance improvements.

In order to maintain security, system administrators must also keep their software up-to-date and respond to the latest TLS protocol vulnerabilities. Fixing the latest vulnerabilities often requires more than a patch. Oftentimes the system administrator has to reconfigure the web server. In the case of Heartbleed, this required updating OpenSSL, revoking the original certificate (which may have required paying more money), generating a new key, revalidating the domain, and retrieving and installing the new certificate.

At the very least, the system administrator must repeat the process before the certificate expires. Most certificates have a validity period of between one and three years, yet many system administrators do not renew their certificates on schedule. In our analysis from Chapter II, we found that 20% of expiring certificates were removed retroactively.

5.2.1 Certificate Marketing

Obtaining a TLS certificate can be a harrowing experience. As a system administrator first entering the market, it can be difficult to even determine which kind of certificate you need. Prices vary so drastically from certificate authority to certificate authority that it can be hard to believe that you are buying the same product. As one example, Symantec, the second largest certificate authority, sells wildcard certificates for $1999 while StartCom offers a functionally identical product for $60 [121, 124]. Certificate authorities rope customers in with rhetoric, snake oil and brand name in order to sell their certificates. Table 5.1 and Table 5.2 show two examples of CAs' different marketing approaches. Symantec, like many other commercial CAs, has a few "value added" features and prominently promotes gimmicks. One of these items, "trust seals," CAs claim fosters greater assurance that the business is trustworthy. "Trust seals" or "SSL seals" are simply images you can display on your site. The image links to the CA website where it will state that the domain is trusted. From a security perspective, trust seals are effectively useless [33, 38]. An attacker can copy

the seal image, place it on their website, and link the seal to an attacker controlled phishing site. A user who is savvy enough to know the exact website a trust seal is supposed to link to would presumably also be savvy enough to simply check the certificate. Another CA tactic is to offer very large monetary warranties. However, these warranties are not for the domain owner, but instead are intended for the end-user [2]. If the end-user suffers fraud from visiting a site signed by the certificate, they are entitled to the warranty money if they can prove negligence by the CA. The warranty can only be collected in cases where the cert requester is an attacker. Legitimate sites do not benefit from purchasing a high-value warranty. Most end-users are completely oblivious to which CA signed the site they are currently viewing and they certainly do not know about the various warranty policies and the amounts.

Often certificate marketing is wrong or misleading. GoDaddy says they offer the "World's Strongest Encryption." "Our SSLs use SHA-2 and 2048-bit encryption to stop hackers in their tracks. That's the strongest encryption on the market today." [16] This level of encryption is actually the minimum allowed by the CAB Forum Baseline Requirements [32]. The statement also implies a false sense of security. Non-cryptographic attacks are extremely prevalent, and the industry standard 2048-bit RSA encryption will not affect such an attack. Symantec also advertises the "strongest security" under the heading of elliptic curve cryptography (ECC). Though ECC is beneficial for its performance and relatively small key size, the strength of the security depends on its curve and key size, and is thought to be comparable to RSA for certain choices of key size.

Symantec takes advantage of the marketing and charges an additional $\sim$ \$600 for ECC's "security". This indicates a lack of market competition. The marginal cost of signing an ECC certificate is the same as signing an RSA certificate; it does not require any additional validation on the part of the CA. An efficient market would charge the same price for these two services. In the current certificate market, CAs sell not only the certificate, but also the branding and imagery; many commercial CAs are in the business of security theater.

Symantec SSL	Secure Site	Secure Site Pro	Secure Site with EV	Secure Site with EV Pro	Secure Site Wildcard
Price	$399	$995	$995	$1499	$1999
Trust Mark	Yes	Yes	Yes	Yes	Yes
ECC: Strongest Security		Yes		Yes	
Warranty	$1,500,000	$1,500,000	$1,750,000	$1,750,000	$1,500,000
Green Address Bar			Yes	Yes	
Critical Vulnerability Scan		Yes	Yes	Yes	

Figure 5.1: **Symantec Price Comparison** — Symantec charges $1500 more for wildcard certificates, which do not require any additional checks. Symantec also charges an additional $600 for ECC certificates. Note: Symantec also has support features that come with every certificate. Prices current as of 10/5/2015 [124]

StartCom	Free	Identity Verified	Organization Verified	Extended Validation
S/MIME Client + Auth	Yes	Yes	Yes	Yes
SSL/TLS Server	Yes	Yes	Yes	Yes
SSL/TLS XMPP	Yes	Yes	Yes	Yes
128/256-Bit Encryption	Yes	Yes	Yes	Yes
Renewable	Yes	Yes	Yes	Yes
Vulnerability Detection	Yes	Yes	Yes	Yes
Multiple Domains (UCC)		Yes	Yes	Yes
Multiple Emails (S/MIME)		Yes	Yes	Yes
Wild Card Capability		Yes	Yes	Yes
Server-Client Authentication		Yes	Yes	Yes
Identification Details		Yes	Yes	Yes
Organization Details			Yes	Yes
Object Code Signing		Yes	Yes	Yes
Time-Stamping				Yes
Microsoft Kernel-Code				Yes
Green Trustbar (EV)				Yes
Validation Level	Class 1	Class 2	Class 2/3	Extended
Certificate Limitations	Unlimited	Unlimited	Unlimited	Unlimited
Certificate Validity	1 Year	2 Years	2/3 Years	2 Years
Price	$0	$59.90	$119.80	$199.90

Figure 5.2: **StartCom Price Comparison** — StartCom avoids marketing speak and uses technically correct terms. Although the description is more verbose, all of the limitations of the certificates are clearly stated. StartCom offers free certificates in some cases, but does charge $25 if the certificate needs to be revoked. Prices current as of 10/5/2015 [121]

This presents another motivation for automating the certificate issuance process. We can more clearly demonstrate the equivalence of the end product and remove the predatory marketing. The lock-icon in the user's browser is the same for every certificate authority. This perfectly competitive end-product should cause CAs to compete on price.

5.2.2 CA Validation

For CAs, there is no incentive to perform any additional checks or take on any additional cost associated with superior security practices. The security of the system is only as good as the security of the least effective CA, as each CA can sign for any domain. Naturally, this causes CAs to race to the bottom in terms of security.

In order to stop the deterioration of the verification practices, tiered validation standards have arisen.

Domain Validation (DV) The CA guarantees only that the certificate requester owns the domain for which they are requesting. The domain is not required to be tied to any real-world identity. From a user perspective, this is aimed to verify that the user is connected to the domain entered into their address bar, but does not protect against phishing attacks.

Organization/Identity Validation (OV) The CA has verified that the certificate requester owns the domain and the requester can be tied to a real-world entity. The CA must verify all additional information located within the subject of the certificate. Unfortunately, from a user perspective, this class of certificate is rarely recognized. It appears the same as a DV certificate within browsers; the certificate can only be recognized by manually investigating the subject field of the certificate.

Extended Validation (EV) Extended validation (EV) was introduced in 2007 as a direct response to the collapsing standards of general certificate verification. EV's goal is

to identify the legal entity that controls a website [12]. EV appears as an additional green bar within browsers, giving users a stronger indicator of trust.

Although the certificate validation tiers exist, the value of them is debatable. Even experts in the field have a difficult time distinguishing between OV and DV certificates [23]. Distinguishing between organization validation and domain validation certificates requires manual inspection. Even the value of EV certificates have been called into question, as users do not necessarily know which sites intend to have EV certificates and users have not been widely educated on the differences [70].

Domain validation is the largest market and represents the baseline for deploying HTTPS on the Internet [23]. As such, we will be primarily focusing on this form of validation.

5.2.3 Existing DV Methods

The current CAB Forum Baseline Requirements do not explicitly restrict how the CA verifies domain names [32]. The CAB Forum's Domain Validation Working Group is attempting to remedy this by formalizing the various techniques. Through reviewing the major commercial CAs practices and the proposed standards, there are three main techniques used to perform online domain validation (DV) for system administrators today [15, 35, 56, 67, 97, 126].

Email Validation By far the most popular method, email validation involves sending a code to a protected email address associated with the domain name. This includes the technical and administrative contacts found in the WHOIS lookup of the domain, along with the following prefixes: `admin`, `administrator`, `postmaster`, `hostmaster`, and `webmaster`. If the certificate requester can provide the token back to the CA, the requester is assumed to have control over the domain.

DNS Validation making a change to information in a DNS record for the Authorization Domain The certificate requester is required to make a change to information in a DNS

record. Exact techniques vary by CA. The CA may require something as simple as publishing a special TXT record, or they may obligate something more complex such as requiring a chosen subdomain to advertise a CNAME record for a CA controlled domain.

HTTP Validation The certificate requester is required to place a text file at a specified location on the web server with a token that is tied to the request. The CA will validate that the file exists in the proper format, which lends evidence to the requester owning the domain.

All three methods are vulnerable to compromise of the DNS system, or MITM attacks of the service. There are two general problems with the validation methods in use today. First, each one of the methods provides slightly different assurances and thereby expands the attack surface. Second, the challenges presented are unique to each CA. This makes it extremely difficult to understand the ramifications for system administrators of different configurations.

Email validation is the most popular method, perhaps because it requires the least amount of technical sophistication. Its security leaves much to be desired. Email validation requires that domains protect and maintain ownership over several email addresses of which they might not be aware. Exacerbating the problem, CAs have been found in clear violation of the approved dictated policies [32, 97] and accept their own unique set of email addresses as authoritative. Given that it is impossible to enumerate over all of the CAs and determine their acceptable validation email prefixes, administrators must protect all email addresses that may infer any sort of authority. There have been numerous documented cases of people attaining certificates for domains they do not own. Zusman famously attained a certificate for `live.com`, a Microsoft owned email service, by registering for the name `SSLCertificates@live.com` [134]. Zusman only had to find one CA out of the thousands that would accept that email as valid in order to attain a certificate.

In addition to requiring the system administrator to protect the accounts on the mail server, email validation may be vulnerable to passive network attacks due to a lack of STARTTLS support on the SMTP server. We found only 52% of SMTP servers supported encryption through STARTTLS [41]. Attackers who are able to view the traffic can respond with the correct token for their session with the CA.

DNS validation is popular amongst web hosting providers, but is seldom used in practice by people running their own web servers. DNS validation aims to prove that the certificate requester currently has control over the domain name system and that the client could direct users to any arbitrary server. In order to be affective, it requires all of the domain's DNS records to be protected from unauthorized modification.

The HTTP validation method verifies that the certificate requester has privileges to add a file to a presumably protected directory on the server at the requested domain. Although this verification method is easily performed and can be automated, this validation method requires that the server administrators maintain proper control over content posted to the server. Mismanaged servers or vulnerabilities that allow attackers to create content in arbitrary locations can circumvent this verification and allow them to attain unauthorized certificates. Since there is not a standardized mechanism, it is impossible to know the exact directories that must be protected.

Given that each CA's authority is flat, i.e. every certificate authority can sign for every domain, the proof of domain ownership should also be agreed upon and regulated. This minimizes the attack surface of the verification process itself, while also allowing concerned parties to know exactly what is expected during verification of domain ownership. The CAB Forum has recently begun discussing more systematic techniques to prove domain ownership [15]. However, these are still being actively revised, have not yet been adopted, and they offer only guidelines.

5.3 ACME

In order to achieve our goals of widespread HTTPS adoption, the entire process of acquiring and installing a certificate must be automated. As we discovered in Section 5.2, the process of domain validation can be completely automated using existing techniques, but no one has automated the full process.

The Automated Certificate Management Environment (ACME) [27] is a protocol we have developed that allows for automated certificate issuance, renewal, and revocation. ACME is a protocol over HTTPS and JSON, making heavy use of JSON Web Signatures [72] for integrity and authentication. For purposes of the protocol, the client is the certificate requester and the server is the certificate authority. The protocol is an IETF Internet draft and is being used by the Let's Encrypt CA [4]. We hope that through standardization we can enable many CAs and client software to interoperate, increasing the relative value of both client development and CA adoption. The latest version of the draft can be found at `https://tools.ietf.org/html/draft-ietf-acme`.

5.3.1 Protocol Overview

ACME allows a client to perform all of the typical certificate management functions using JSON messages over HTTPS. ACME is meant to mimic a traditional CA, in which a user creates an account, authorizes identifiers (domains) with the account, and requests certificate issuance.

ACME accounts are represented as a public/private key pair, referenced as the account key. Identifiers are added to an account by authorizing the account key for a given domain. Certificate issuance and revocation are authorized by a signature with the account key.

The first phase of ACME is for the client to register with the ACME server. The client generates an asymmetric key pair (the account key) and associates this key pair with contact information by signing it. The server acknowledges the registration by replying with a registration object echoing the client's input.

Before a client can issue certificates, it must establish an authorization with the server for an account key pair to act for the identifier(s) that it wishes to include in the certificate. To do this, the client must demonstrate to the server both (1) that it holds the private key of the account key pair, and (2) that it has authority over the identifier being claimed.

Proof of possession of the account key is built into the ACME protocol. All messages from the client to the server are signed by the client, and the server verifies them using the public key of the account key pair.

To verify that the client controls the identifier being claimed, the server issues the client a set of challenges. Because there are many different ways to validate possession of different types of identifiers, the server will choose from an extensible set of challenges that are appropriate for the identifier being claimed. The client responds with a set of responses that tell the server which challenges the client has completed. The server then validates the challenges to check that the client has achieved the authorization.

Once the client has authorized an account key pair for an identifier, it can use the key pair to authorize the issuance of certificates for the identifier. The client sends a PKCS#10 Certificate Signing Request (CSR) to the server (indicating the identifier(s) to be included in the issued certificate) and a signature over the CSR by the private key of the account key pair.

If the server agrees to issue the certificate, then it creates the certificate and provides it in its response. The certificate is assigned a URI, which the client can use to fetch updated versions of the certificate.

Revocation is performed by having the client send a revocation request signed by one of two keys, either the current account key that achieved authorization over the identifiers, or the private key contained within the certificate. The server indicates whether the request has succeeded.

ACME is defined with enough flexibility to handle different types of identifiers, but the primary use case addressed by ACME is where domain names are used as identifiers. The

use of ACME for other protocols will require further specification, in order to describe how these identifiers are encoded in the protocol and what types of validation challenges the server might require.

Next, we will present how the protocol is structured and then describe the specific messages.

5.3.2 Protocol Elements

ACME is structured as a RESTful protocol [53]. Each ACME function is accomplished by the client sending a sequence of HTTPS requests to the server carrying JSON messages.

All ACME requests with a non-empty body are encapsulated in a JSON Web Signature (JWS) object [72], signed using the account key pair. The server verifies the JWS before processing the request. Encapsulating request bodies in JWS provides a simple authentication of requests by way of key continuity.

Note that this implies that GET requests are not authenticated. Servers cannot respond to GET requests for resources that might be considered sensitive.

An ACME request carries a JSON dictionary that provides the details of the client's request to the server. In order to avoid attacks that might arise from sending a request object to a resource of the wrong type, each request object has a "resource" field that indicates what type of resource the request is addressed to, as defined in Table 5.1.

Resource type	"resource" value
New registration	new-reg
Recover registration	recover-reg
New authorization	new-authz
New certificate	new-cert
Revoke certificate	revoke-cert
Registration	reg
Authorization	authz
Challenge	challenge
Certificate	cert

Table 5.1: **ACME Resources** — ACME resources and their resource values.

For the "new-X" resources in Table 5.1, the server can only have one resource for each function. This resource may be addressed by multiple URIs, but all must provide equivalent functionality.

ACME uses different URIs for different management functions. Each function is listed in a directory, along with its corresponding URI, so clients only need to be configured with the directory URI.

The "up" link relation is used with challenge resources to indicate the authorization resource to which a challenge belongs. It is also used with certificate resources to indicate a resource from which the client may fetch a chain of CA certificates that could be used to validate the certificate in the original resource.

Figure 5.3 illustrates the relations between resources on an ACME server.

Action	Request	Response
Request Challenges	POST new-reg	201 → reg
Answer Challenges	POST new-authz	201 → authz
Poll for Status	GET authz	200
Request Issuance	POST new-cert	201 → cert
Check for New Cert	GET cert	200

Table 5.2: **Expected ACME Flow** — Requests and Responses are over HTTPS. "→" is a mnemonic for a Location header pointing to a created resource.

Table 5.2 illustrates a typical sequence of requests required to establish a new account with the server, prove control of an identifier, issue a certificate, and fetch an updated certificate some time after issuance.

The remainder of this section provides the details of how these resources are structured and how the ACME protocol makes use of them.

5.3.3 Directory

In order to help clients configure themselves with the right URIs for each ACME operation, ACME servers provide a directory object. This should be the root URL with

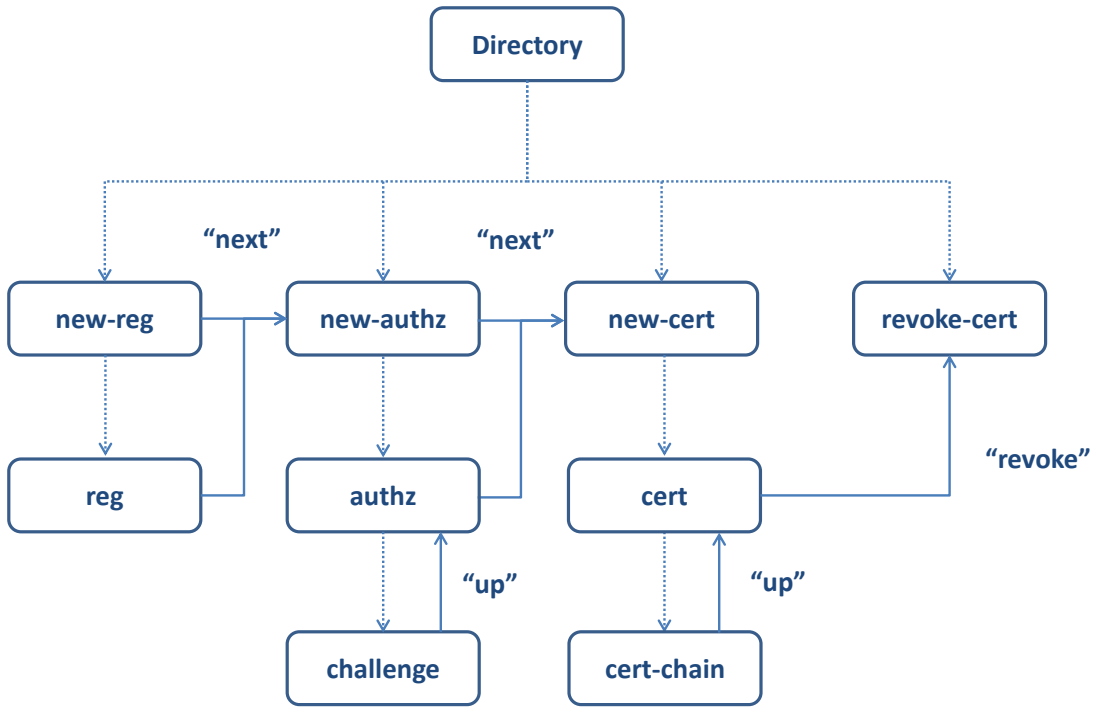

Figure 5.3: **ACME Resource Relationships** — This figure illustrates the relations between resources on an ACME server. The solid lines indicate link relations, and the dotted lines correspond to relations expressed in other ways, e.g., the Location header in a 201 (Created) response

which clients are configured. It is a JSON dictionary, with keys that are the "resource" values listed in Table 5.1 and with values that are the URIs used to accomplish the corresponding function.

Clients access the directory by sending a GET request to the directory URI. Once the client is configured, it can proceed onto registration.

5.3.4 Registration

An ACME registration resource represents a set of metadata associated with an account key pair. Registration resources have the following structure:

key (required, dictionary): The public key of the account key pair, encoded as a JSON Web Key (JWK) object [71].

contact (optional, array of string): An array of URIs that the server can use to contact the client for issues related to this authorization. For example, the server may wish to notify the client about server-initiated revocation.

agreement (optional, string): A URI referring to a subscriber agreement or terms of service provided by the server. Including this field indicates the client's agreement with the referenced terms.

authorizations (optional, string): A URI from which a list of authorizations granted to this account can be fetched via a GET request. The result of the GET request is a JSON object whose "authorizations" field is an array of strings, where each string is the URI of an authorization belonging to this registration. The server includes pending authorizations, and does not include authorizations that are invalid or expired.

certificates (optional, string): A URI from which a list of certificates issued for this account can be fetched via a GET request. The result of the GET request is a JSON object whose "certificates" field is an array of strings, where each string is the URI of a valid certificate.

5.3.5 Authorization

An ACME authorization object represents an account's authorization over an identifier. An authorization object includes the identifier, which challenges are required or were used to attain authorization, as well as several metadata fields.

identifier (required, dictionary of string): The identifier associated with the authorization. The identifier must have the following two fields:

type (required, string): The type of identifier. In the case of HTTPS, this is `dns`.

value (required, string): The identifier itself.

status (optional, string): The status of this authorization. Possible values are: `unknown`, `pending`, `processing`, `valid`, `invalid`, and `revoke`. If this field is missing, then the default value is `pending`.

expires (optional, string): The date after which the server will consider this authorization invalid, encoded in the format specified in RFC 3339 [76].

challenges (required, array): The challenges that the client needs to fulfill in order to prove possession of the identifier (for pending authorizations). For final authorizations, the challenges that were used. Each array entry is a dictionary with parameters required to validate the challenge.

combinations (optional, array of arrays of integers): A collection of sets of challenges, each of which would be sufficient to prove possession of the identifier. Clients complete a set of challenges that covers at least one set in this array. Challenges are identified by their indices in the challenges array. If no `combinations` element is included in an authorization object, the client completes all challenges.

All of the information contained within the Authorization resource is designed to be public information in order to allow the CA to publish it for transparency purposes. Section 5.4 describes and compares ACME challenges for `dns` identifiers.

5.3.6 Certificate Issuance

The holder of an authorized key pair for an identifier may use ACME to request that a certificate be issued for that identifier. The client makes this request by sending a POST request to the server's new-certificate resource. The body of the POST is a JWS object whose JSON payload contains a Certificate Signing Request (CSR) [105]. The CSR encodes the parameters of the requested certificate; authority to issue is demonstrated by the JWS signature with an account key, from which the server can look up related authorizations.

The new-cert request contains only the CSR.

csr (required, string): A CSR encoding the parameters for the certificate being requested. The CSR is sent in the Base64-encoded version of the DER format.

The CSR encodes the client's requests with regard to the content of the certificate to be issued. Of course, the values provided in the CSR are only a request and are not guaranteed. The server or CA may alter any fields in the certificate before issuance. For example, the CA may remove identifiers that are not authorized for the account key that signed the request.

It is up to the server's local policy to decide which names are acceptable in a certificate, given the authorizations that the server associates with the client's account key. For instance, many CAs certify wildcard certificates after verifying the underlying domain name, without the "*" DNS label. Future ACME servers may consider that client authorized for a wildcard domain. It is important, though, that servers not extend authorization across identifier types. Just because a client is authorized for `example.com`, does not mean the client has control over the IP address that `example.com` points to. CAs must authorize the identities contained within the certificate.

If the CA decides to issue the certificate, the server will create a new certificate resource and return a URI for it in the Location header field of a 201 (Created) response.

It may also include the certificate in the body of the response, if it is available. Generally though, the client should retrieve the certificate with a GET request to the certificate URI and poll for the certificate. If the certificate still isn't available, the server will provide a 202 (Accepted) response and include a Retry-After header to indicate when the server believes the certificate will be issued. By default, the certificates are encoded in DER, though the client can request other formats by including an Accept header in the request.

Additionally, in the certificate response, the server provides metadata about the certificate in the HTTP headers. In particular, the server will include a Link relation header field [103] with relation "up" to provide the certificate immediately preceding it in the certificate chain. In order to aid implementations, the certificate resource also contains an "author" relation to indicate which registration object the certificate was issued.

Certificate resources always represent the most recent certificate issued for the name/key binding expressed in the CSR. If the CA allows a certificate to be renewed, then it publishes renewed versions of the certificate through the same certificate URI.

Clients retrieve renewed versions of the certificate using a GET query to the certificate URI, which the server should then return in a 200 (OK) response. The server provides a stable URI for each specific certificate in the Content-Location header field.

To avoid unnecessary renewals, the CA may choose not to issue a renewed certificate until it receives such a request (if it allows renewal at all). In such cases, if the CA requires some time to generate the new certificate, the CA will return a 202 (Accepted) response, with a Retry-After header field that indicates when the new certificate will be available. The CA may include the current (non-renewed) certificate as the body of the response.

This does present an opportunity for unauthorized parties to prompt unnecessary renewals, thus the URIs should be structured as capability URLs [125].

Clients do not need to know whether a certificate URI allows renewals. If the client's GET request to the URI doesn't yield an updated certificate, the client can initiate a new-certificate transaction to request one.

5.3.7 Revocation

Revocation is performed by the client sending a POST request to the ACME servers revoke-cert URI. The body of the POST is a JWS object whose JSON payload contains the certificate to be revoked:

certificate (required, string): The certificate to be revoked, in the Base64-encoded version of the DER format. Note that this is not PEM, as we use a URI safe Base64-encoding throughout ACME.

Revocation requests in ACME are slightly different from other ACME requests in that they can be validly signed with two different keys. It may, of course, be signed with the

current account key, one that has achieved authorization over all of the identifiers in the certificate at some point. It may also be signed by the private key of the certificate itself, as the client will have demonstrated complete control over the certificate.

If the signature is valid, and the server accepts the request, it responds with status code 200 (OK). If the revocation fails, the server returns an error.

5.3.8 Account Recovery

Once a client has created an account with an ACME server, it is possible that the private key for the account will be lost. The recovery contacts included in the registration allows the client to recover from this situation, as long as it still has access to these contacts.

By "recovery," we mean that the information associated with an old account key is bound to a new account key. When a recovery process succeeds, the server provides the client with a new registration whose contents are the same as the base registration object—except for the "key" field, which is set to the new account key. The server reassigns resources associated with the base registration to the new registration (e.g., authorizations and certificates). The server should delete the old registration resource after it has been used as a base for recovery.

In addition to the recovery mechanisms defined by ACME, individual client implementations may also offer implementation-specific recovery mechanisms. For example, if a client creates account keys deterministically from a seed value, then this seed could be used to recover the account key by re-generating it. Or an implementation could escrow an encrypted copy of the account key with a cloud storage provider, and give the encryption key to the user as a recovery value.

When implementing any recovery mechanism, it is important to remember that ACME's security is only as good as its weakest link. The security of the particular adopted medium must be fully analyzed.

5.3.8.1 Contact-Based Recovery

In the contact-based recovery process, the client requests that the server send a message to one of the contact URIs registered for the account. That message indicates some action that the server requires the client's user to perform, e.g., clicking a link in an email. If the user successfully completes the server's required actions, then the server will bind the account to the new account key.

(Note that this process is almost entirely out of band with respect to ACME. ACME only allows the client to initiate the process, and the server to indicate the result.)

To initiate contact-based recovery, the client sends a POST request to the server's recover-registration URI, with a body specifying which registration is to be recovered. The body of the request is signed by the client's new account key pair.

method (required, string): The string "contact"

base (required, string): The URI for the registration to be recovered.

If the server agrees to attempt contact-based recovery, then it creates a new registration resource containing a stub registration object. The stub registration has the client's new account key and contacts, but no associated authorizations or certificates. The server returns the stub contact in a 201 (Created) response, along with a Location header field indicating the URI for the new registration resource (which will be the registration URI if the recovery succeeds).

After recovery has been initiated, the server follows its chosen recovery process, out-of-band to ACME. While the recovery process is ongoing, the client may poll the registration resource's URI for status, by sending a POST request with a trivial body ("resource":"reg"). If the recovery process is still pending, the server sends a 202 (Accepted) status code, and a Retry-After header field. If the recovery process has failed, the server sends an error code (e.g., 404), and deletes the stub registration resource.

If the recovery process has succeeded, then the server will send a 200 (OK) response, containing the full registration object, with any necessary information copied from the old registration. The client may now use this in the same way as if he had attained it from a new-registration transaction.

5.3.9 Security and Considerations

ACME is a security protocol designed to verify public keys belonging to the identified domains. Therefore, insuring the integrity of the process is of utmost importance. Specifically, ACME must verify that only entities with control over identifiers (domains) can achieve authorization for the identifier. Once an identifier is authorized under an account key, it must not be possible to improperly transfer the authorization to another account key.

In this section, we will discuss the threat model and possible attacks. We will describe how ACME achieves its outlined security goals under the proposed threat model.

5.3.9.1 Threat Model

The ACME protocol is performed over three channels.

1. The original ACME channel, the HTTPS channel used to send ACME messages.

2. A domain validation channel, the channel in which challenges are performed to verify the domain.

3. The contact channel, the channel used to contact the ACME registrar used in account recovery.

An overview of the channels can be seen in Figure 5.4.

In practice, the risks to these channels are not entirely separate, but they are different in most cases. Each of the three channels, for example, uses a different communications pattern: the ACME channel will comprise inbound HTTPS connections to the ACME server,

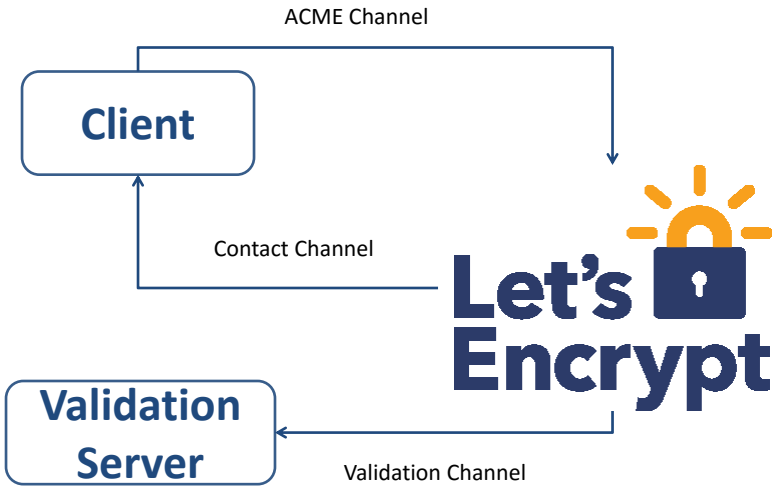

Figure 5.4: **ACME Channels**

the validation channel outbound HTTP or DNS requests, and the contact channel will use channels such as email and PSTN.

ACME has been designed to be resilient to passive and active attackers on any individual channel. The protocol has also been designed to be resistant to application-layer MITM attacks. This has the benefit of allowing ACME CAs the ability to use TLS termination CDN services, such as CDNs, without having to worry about the intentions or security of the middleboxes. In addition to the protocol having built-in resistance, ACME does recommend clients support HTTP public key pinning [118], and that servers emit pinning headers.

Next, we will describe ACME's anti-replay mitigation. The full analysis of authorizations and their security can be found in Section 5.4.

5.3.9.2 Replay Protection

As malicious middleboxes are in the ACME's threat model, replay protection must be provided. ACME requests have a mandatory anti-replay mechanism. This mechanism is

based on the server maintaining a list of nonces that it has issued to clients, and requires any signed request from the client carries such a nonce.

An ACME server must include a Replay-Nonce header field in each successful response it provides to a client, with contents as specified below. In particular, the ACME server provides a Replay-Nonce header field in response to a HEAD request for any valid resource. (This allows clients to easily obtain a fresh nonce.) It MAY also provide nonces in error responses.

Every JWS sent by an ACME client must include, in its protected header, the "nonce" header parameter, with contents as defined below. As part of JWS verification, the ACME server must verify that the value of the"nonce" header is a value that the server previously provided in a Replay-Nonce header field. Once a nonce value has appeared in an ACME request, the server must consider it invalid, just as a value it had never issued.

When a server rejects a request because its nonce value was unacceptable (or not present), it will provide an HTTP status code of 400 (Bad Request), and indicate the ACME error code "urn:acme:badNonce".

The precise method used to generate and track nonces is up to the server. For example, the server could generate a random 128-bit value for each response, keep a list of issued nonces, and strike nonces from this list as they are used.

The "Replay-Nonce" header field includes a server-generated value that the server can use to detect unauthorized replay in future client requests. The server should generate the value provided in Replay-Nonce in such a way that they are unique to each message, with high probability.

The value of the Replay-Nonce field must be an octet string, encoded according to the base64url encoding [72]. Clients must ignore invalid Replay-Nonce values.

The "nonce" header parameter provides a unique value that enables the verifier of a JWS to recognize when replay has occurred. The "nonce" header parameter must be carried in the protected header of the JWS.

The value of the "nonce" header follows the same encoding as the "Replay-Nonce" header. If the header value does not follow the correct encoding, the server must reject the JWS as malformed.

5.3.9.3 ACME and Authorizations

Anyone can register an account key with the ACME server, but they should be unable to prove ownership of the domain over the validation channel with the account key. Thus, all authorizations must guarantee that the identifier owner is in control over the validation, and that the validation is attached to the account key for which the challenge was issued. Challenges and how they address the threat model are discussed in Section 5.4.

Essentially, each challenge is tied directly to the account key for which the challenge was issued. All three of the challenges utilize "authorized key objects" which contain both the token (specific to the challenge) and the account public key. Thus, the validation channel is tied both to the account and the particular session within the ACME channel.

The final step that must be guaranteed is the integrity of the account keys themselves. It should not be possible to transfer authorizations from one account to another outside of the account recovery mechanisms. Every change of state within ACME is signed by the owner's account key, except for the account recovery key and contact mechanisms.

5.4 Identifier Validation Challenges

The cornerstone of the protocol is the set of challenges which prove the account holder also owns the identifier in question. Traditionally, CAs have relied on a variety of means to test whether an entity applying for a certificate with a given identifier actually controls that identifier. ACME attempts to standardize mechanisms that do not require human interaction or require minimal interaction. For each type of mechanism or challenge, in order for an entity to successfully complete the challenge, the entity must both:

- Hold the private key of the account key pair used to respond to the challenge
- Control the identifier in question

In order to ensure these properties, ACME includes an extensible challenge/response framework for identifier validation. The general structure of Challenge payloads is as follows:

type (required, string): The type of Challenge encoded in the object.

uri (required, string): The URI to which a response can be posted.

status (optional, string): The status of this authorization. The possible values are: "unknown," "pending," "processing," "valid," "invalid," and "revoked."

validated (optional, string): The time at which this challenge was completed by the server, encoded in the format specified in RFC 3339. [76]

error (optional, dictionary of strings): The error that occurred while the server was validating the challenge, if any. This field is structured as a problem document [104].

Different challenges allow the server to obtain proof of different aspects of control over an identifier. In some challenges, like Simple HTTP and DVSNI, the client directly proves its ability to do certain things related to the identifier. In the Proof of Possession challenge, the client proves historical control of the identifier, by reference to a prior authorization transaction or certificate.

The choice of which challenges to offer to a client under which circumstances is a matter of server policy. A CA may choose different sets of challenges depending on whether it has interacted with a domain before, and how. For example:

- New domain with no known certificates: Domain Validation (DVSNI or Simple HTTP)

- Domain for which known certificates exist from other CAs — DV + Proof of Possession of previous CA-signed key
- Domain with a certificate from this CA, lost account key — DV + PoP of ACME-certified Subject key
- Domain with a certificate from this CA, all keys and recovery mechanisms lost — out of band proof of authority for the domain

The identifier validation challenges described in this section all relate to validation of domain names. If ACME is extended in the future to support other types of identifiers, new challenge types will need to be defined. Challenges will need to specify which identifiers they apply to.

Next, we will describe the domain ownership challenges. The goal of these challenges is to confirm the client has authoritative control over the domain in question. The SimpleHTTP and DNS challenges are similar to commonly used techniques today, while the DVSNI challenge is a novel technique and provides slightly stronger assurances.

The domain ownership challenges all have the same goal, to verify control of the identifier and prove the client also controls the private key of the account key pair. In order to simplify and ease implementation and analysis of the protocol, all of the challenges make use of an "authorized key" object. Such an object is a JSON object that encodes an authorization for a specific account key to fulfill a specific challenge. An authorized key object consists of two fields.

token (required, string): A random value that uniquely identifies a challenge. This value must have at least 128 bits of entropy, in order to prevent an attacker from guessing it. It cannot contain any characters outside the URL-safe Base64 alphabet.

key (required, JWK): The account key being authorized.

5.4.1 SimpleHTTP

SimpleHTTP provides the typical, upload a file to your web server, validation challenge utilized by current CAs. As a domain may resolve to multiple IPv4 and IPv6 addresses, the server will connect to at least one of the hosts found in A and AAAA records, at its discretion. The HTTP server may be made available over either HTTPS or unencrypted HTTP; the client tells the server in its response which to check.

type (required, string): The string "simpleHttp"

authorizedKey (required, string): A serialized authorized key object, base64-encoded. The "key" field in this object must match the client's account key.

A client responds to this challenge by parsing the authorized key object, verifying that its "key" field contains the client's account key, and provisioning it as a resource on the HTTP server for the domain in question. The path at which the resource is provisioned is comprised of the fixed prefix `.well-known/acme-challenge`, followed by the "token" value in the challenge.

The client's response to this challenge indicates its agreement.

type (required, string): The string "simpleHttp"

token (required, string): The "token" value from the authorized key object in the challenge.

Given a Challenge/Response pair, the server verifies the client's control of the domain by verifying that the resource was provisioned as expected.

1. Verify that the "token" value in the response matches the "token" field in the authorized key object in the challenge.

2. Form a URI by populating the URI template [60] `http://{domain}/.well-known/acme-challenge/{token}` where:

- the domain field is set to the domain name being verified; and
- the token field is set to the token in the authorized key object.

3. Verify that the resulting URI is well-formed.
4. Dereference the URI using an HTTP GET request.
5. Verify that the Content-Type header of the response is either absent, or has the value "application/json".
6. Verify that the body of the response is a well-formed authorized key object.
7. Verify that the "key" and "token" fields in the authorized key object match the values from the authorized key object in the challenge.

Comparisons of the "token" field must be performed in terms of Unicode code points, taking into account the encodings of the stored nonce and the body of the request.

If all of the above verifications succeed, then the validation is successful. If the request fails, or the body does not pass these checks, then it has failed.

5.4.2 DVSNI

The Domain Validation with Server Name Indication (DVSNI) validation method proves control over a domain name by requiring the client to configure a TLS server referenced by an A/AAAA record under the domain name. The server must respond to specific connection attempts utilizing the Server Name Indication extension [45]. The server verifies the client's challenge by accessing the reconfigured server and verifying a particular challenge certificate is presented.

The DVSNI challenge has the following form.

type (required, string): The string "dvsni"

authorizedKey (required, string): A serialized authorized key object, base64-encoded. The "key" field in this object matches the client's account key.

n (required, number): Number of DVSNI iterations

In response to the challenge, the client must decode and parse the authorized key object and verify that it contains exactly one entry, whose "token" and "key" attributes match the token for this challenge and the client's account key. The client then computes the SHA-256 digest Z_0 of the JSON-encoded authorized key object (without base64-encoding), and encodes Z_0 in UTF-8 lower-case hexadecimal form. The client then generates iterated hash values $Z_1 ... Z_{n-1}$ as follows:

$$Z_i = lowercase\_hexadecimal(SHA256(Z_{i-1}))$$

The client generates a self-signed certificate for each iteration of Z_i with a single subjectAlternativeName extension dNSName that is $Z_i[0:32].Z_i[32:64]$.acme.invalid, where $Z_i[0:32]$ and $Z_i[32:64]$ represent the first 32 and last 32 characters of the hex-encoded value, respectively (following the notation used in Python). The client then configures the TLS server at the domain such that when a handshake is initiated with the Server Name Indication extension set to $Z_i[0:32].Z_i[32:64]$.acme.invalid, the corresponding generated certificate is presented.

When the client is ready, it simply acknowledges the challenge by sending the challenge type and token back to the challenge URI.

type (required, string): The string "dvsni"

token (required, string): The "token" value from the authorized key object in the challenge.

Given a Challenge/Response pair, the ACME server verifies the client's control of the domain by verifying that the TLS server was configured appropriately.

1. Verify that the "token" value in the response matches the "token" field in the authorized key object in the challenge.

2. Choose a subset of the N DVSNI iterations to check, according to local policy.

3. For each iteration, compute the Z_i value from the authorized key object in the same manner as the client.

4. Open a TLS connection to the domain name being validated on the requested port, presenting the value $Zi[0:32].Zi[32:64]$.acme.invalid in the SNI field (where the comparison is case-insensitive).

5. Verify that the certificate contains a subjectAltName extension with the dNSName of $Z[0:32].Z[32:64]$.acme.invalid, and that no other dNSName entries of the form "*.acme.invalid" are present in the subjectAltName extension.

The ACME server should verify a random subset of the N iterations to ensure that an attacker who can provision certificates for a default virtual host, but not for arbitrary simultaneous virtual hosts, cannot pass the challenge. For instance, testing a subset of 5 of $N = 25$ domains ensures that such an attacker has only a one in $25!/(25-5)!$ chance of success if they post certificates Z_j in random succession and happened to have the ability to change the certificate between each request. (This probability is enforced by the requirement that each certificate have only one Z_i value.)

If all of the above verifications succeed, then the validation is successful. Otherwise, the validation fails.

5.4.3 DNS

When the identifier being validated is a domain name, the client can prove control of that domain by provisioning resource records under it. The DNS challenge requires the client to provision a TXT record containing a designated value under a specific validation domain name.

type (required, string): The string “dns”

authorizedKey (required, string): A serialized authorized key object, base64-encoded. The “key” field in this object must match the client’s account key.

Similar to the DVSNI challenge, the DNS challenge also requires the client parse the authorized key object and verify that its “key” field contains the client’s account key. The client then computes the SHA-256 digest of the JSON-encoded authorized key object (without base64-encoding).

The record provisioned to the DNS is the base64 encoding of this digest. The client constructs the validation domain name by prepending the label `_acme-challenge` to the domain name being validated, then provisions a TXT record with the digest value under that name. For example, if the domain name being validated is `example.com`, then the client would provision the following DNS record: `_acme-challenge.example.com. 300 IN TXT ‘‘gfj9Xq...Rg85nM’’`

Similar to the DVSNI challenge, the DNS challenge response simply acknowledges that the client is ready.

type (required, string): The string “dns”

token (required, string): The “token” value from the authorized key object in the challenge.

To validate a DNS challenge, the server performs the following steps:

1. Verify that the “token” value in the response matches the “token” field in the authorized key object in the challenge.
2. Compute the SHA-256 digest of the authorized key object
3. Query for TXT records under the validation domain name
4. Verify that the contents of one of the TXT records matches the digest value

If all of the above verifications succeed, then the validation is successful. If no DNS record is found, or the DNS record and response payload do not pass these checks, then the validation fails.

5.4.4 Proof of Possession

The Proof of Possession challenge verifies that a client possesses a private key corresponding to a server-specified public key, as demonstrated by its ability to sign with that key. This challenge is meant to be used when the server knows of a public key that is already associated with the identifier being claimed, and wishes for new authorizations to be authorized by the holder of the corresponding private key. For DNS identifiers, for example, this can help guard against domain hijacking.

This method is useful if a server policy calls for issuing a certificate only to an entity that already possesses the subject private key of a particular prior related certificate (perhaps issued by a different CA). It may also help enable other kinds of server policies that are related to authenticating a client's identity using digital signatures.

This challenge proceeds in much the same way as the proof of possession of the authorized key pair in the main ACME flow (challenge + authorizationRequest). The server provides a nonce and the client signs over the nonce. The main difference is that rather than signing with the private key of the key pair being authorized, the client signs with a private key specified by the server. The server can specify which key pair(s) are acceptable directly (by indicating a public key), or by asking for the key corresponding to a certificate.

The server provides the following fields as part of the challenge:

type (required, string): The string "proofOfPossession"

certs (optional, array of string): An array of certificates, in Base64-encoded DER format, that contain acceptable public keys.

In response to this challenge, the client uses the private key corresponding to one of the

acceptable public keys to sign a JWS object, including data related to the challenge. The validation object covered by the signature has the following fields:

type (required, string): The string "proofOfPossession"

identifiers (required, identifier): A list of identifiers for which the holder of the prior key authorizes the new key

accountKey (required, JWK): The client's account public key

This JWS is not required by the protocol to have a "nonce" header parameter (as with the JWS objects that carry ACME request objects). This allows proof-of-possession response objects to be computed off-line. For example, as part of a domain transfer, the new domain owner might require the old domain owner to sign a proof-of-possession validation object, so that the new domain owner can present that in an ACME transaction later.

The validation JWS contains a "jwk" header parameter indicating the public key under which the server should verify the JWS.

The client's response includes the server-provided nonce, together with a signature over that nonce by one of the private keys requested by the server.

type (required, string): The string "proofOfPossession"

authorization (required, JWS): The validation JWS

To validate a proof-of-possession challenge, the server performs the following steps:

1. Verify that the public key in the "jwk" header of the "authorization" JWS corresponds to one of the certificates in the "certs" field of the challenge
2. Verify the "authorization" JWS using the key indicated in its "jwk" header
3. Decode the payload of the JWS as UTF-8 encoded JSON
4. Verify that there are exactly three fields in the decoded object, and that:

- The “type” field is set to “proofOfPossession”
- The “identifier” field contains the identifier for which authorization is being validated
- The “accountKey” field matches the account key for which the challenge was issued

If all of the above verifications succeed, then the validation is successful. Otherwise, the validation fails.

5.4.5 Comparing Challenges

In this section, we will review the advantages and disadvantages of the challenges and how they compare to existing CA practices. Each mechanism’s primary goal is to provide an accurate testament that the client and the domain owner are the same identity. Thus, we must analyze how well the challenges achieve this goal, while also considering the ease of adoption of the challenges, and which infrastructure is required to perform the challenge.

Considering the security of the challenges, there are two general classes of attacks that must be scrutinized. The first class of attacks are network attacks. Network attacks can be categorized into “passive” and “active” attacks. Passive attacks have the ability to listen on the connection, but do not have the ability to modify the traffic content. Active network attacks, on the other hand, have the ability to both listen and modify traffic. All widely deployed and proposed domain validation techniques are potentially vulnerable to network attacks, as each is being completely performed over the Internet. In ACME, these attacks are mitigated at the CA policy and protocol level.

One mitigation against network attacks is the recommendation to use multi-path probing techniques for all domain validation challenges (SimpleHTTP, DVSNI, and DNS). Multi-path probing for server authentication involves connecting to the server from many different geographic locations. This technique for detecting authentic servers relies on the fact that

most man-in-the-middle (MITM) attacks or active attacks are local, confined to a specific path to the server. By connecting from many different locations and confirming that the same results are received from each, you substantially raise the bar for the attacker as they must now intercept the connection from "all-sides".

In addition, ACME CAs should also check the DNSSEC status of DNS records used in ACME validation (for zones that are DNSSEC enabled) and apply mitigations against DNS off-path attackers. For instance, CAs can add entropy to their DNS requests [128] or use TCP.

The second class of attacks are on the specific deployment medium of the challenges. It is important to determine which privileges and resources are being relied upon to be strictly controlled by the domain owner. The SimpleHTTP and DVSNI challenges are both meant to demonstrate control over the web server for which the domain points. Ideally, the server would speak a different protocol on the target port, essentially making it execute an arbitrary action that the server would not perform unless the administrator had full administrative access over the port/server.

One potential problem of validating on specific ports, 80 and 443 specifically, is the potential asymmetry in configuration of the web server. It is common for multiple domain names to be directed to the same IP address, but to offer different services at the various ports. Perhaps the most common problem is in shared hosting environments where there are several HTTP hosts, but only one HTTPS host. Both Apache and Nginx will serve a default host if the incoming connection doesn't match any of their expected domains. If the system administrator has not setup a default host themselves, it will default to one of the clients. The challenges should be architected such that these shared web servers do not allow a client to get a certificate for another domain for which they share a server.

SimpleHTTP attempts to use an administrative directory, which should require administrative rights if the server is optimally configured. The CABForum is currently in the process of further standardizing the HTTP challenge, which should further bolster adoption

of protecting the *.well-known* directory by default. The fear is that the server may have a vulnerability that allows the attacker to write arbitrary files to the web server, bypassing the requirement of administrator rights. This is a common problem for web applications. SimpleHTTP assumes that all domains being validated will respond to requests on port 80. Thus, the SimpleHTTP challenge is vulnerable to the default host problem only if the domain is not serving any traffic on port 80. This is more uncommon than domains not serving content on port 443.

DVSNI guarantees administrative privilege over the webserver. The client is forced to modify the existing configuration and serve certificates for invalid domain names. This requires that the user have permission over the server and is able to serve completely arbitrary content. DVSNI requires the client to setup many certificates to be verified in order to avoid the default host problem. If the number of iterations and required checks is chosen appropriately, it should be infeasible for an attacker with only partial control of the webserver to attain a certificate.

The DNS challenge relies on the fact that non-administrative users do not typically need access to DNS records. By having restricted access, and being managed often by a completely different entity, the strength of the challenge depends on the security practices of the DNS provider.

There is one final ACME stopgap that helps thwart both network attacks and attacks on the challenge medium. A CA server policy can be adopted, requiring an additional proof-of-possession challenge for particular identifiers. ACME servers can potentially protect known certificate holders from issuing a certificate for an attacker who has temporary control to perform a DV challenge on the target domain. If the domain already has a certificate, either from the ACME CA, found in a certificate transparency log, or found in our public `scans.io` dataset, the CA can require a proof-of-possession for the key in the earlier certificate. This gives ACME a trust-on-first-use property.

On the usability front, each one of the challenges has a slightly different audience.

SimpleHTTP makes it extremely easy to get a certificate for your domain if you have shared hosting and only have access to the file system; SimpleHTTP can be solved manually. DVSNI works easily with users that have automated clients, while DNS will be more often be utilized by web hosting companies that want certificates for all of the domains that they manage.

5.5 Additional ACME Benefits

ACME enables many different enhancements to the current PKI. In particular, it has the opportunity to solve the problems with revocation through short-lived certificates. Future work found in Chapter VI also details how ACME can solve the trust-agility problem and the anti-competitive marketplace.

5.5.1 Short-Lived Certificates

There are several reasons why certificates may need to be revoked. The certificates may be misissued, the private keys could become compromised, a massive vulnerability (e.g. Heartbleed) may have left keys vulnerable, or the certificate creation may have been a mistake. Unfortunately, revocation has long been considered broken.

There are several reasons why both CRLs and OCSP fail to meet their intended goals. For one, systems relying on online checks tend to "soft-fail," or treat the certificate as trusted when the revocation server cannot be reached. This becomes a problem when using HTTPS because a man-in-the-middle (MITM) attacker is usually also able to block the connection to the revocation server or reply with a "tryLater" response. [87] If client applications did "hard-fail" when unable to contact revocation servers, it would present a single point of failure for all reliant HTTPS connections and any downtime would be catastrophic. Even if the servers did have acceptable availability, current networking implementations, like captive portals, create false positives.

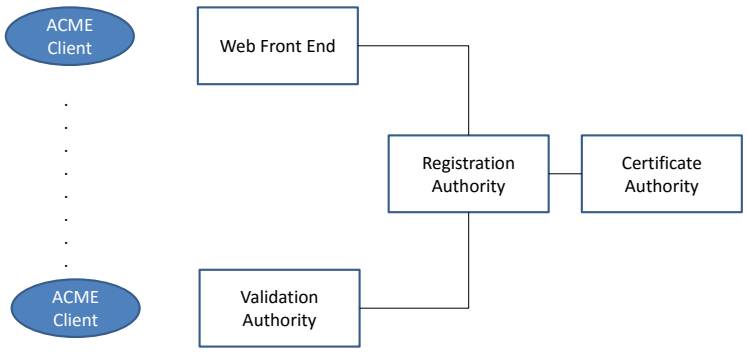

Figure 5.5: **Boulder Organization** — This figure gives a general outline of how the Boulder components and modules are connected.

Even if the attacker is unable to MITM the connection to the OCSP server, there are still attacks involving OCSP stapling they can perform to bypass the active check. [79]

One way to get around the need for revocation is to have short-lived certificates. The certificates become invalid after a short period of time naturally. The downside of short-lived certificates is that they have previously been considered too burdensome to maintain. ACME changes this dynamic with tight integration and automatic reissuance. Let's Encrypt will begin issuing certificates with a 90-day expiry, but we hope to decrease the lifetimes further once clients are better established.

5.6 Implementations

In this section, we will discuss the Let's Encrypt implementations of ACME. Let's Encrypt supports both a CA implementation and an extensible client. The implementations are open-source and available on GitHub [68].

5.6.1 Boulder CA

Boulder is an ACME server written in Go that will be used by Let's Encrypt to issue certificates. Go was the language chosen, as security and performance are of the greatest importance for the server. Go happens to excel at both. Boulder is broken up into logical components: a web front end, registration authority, validation authority, storage authority, and certificate authority. The component organization is shown in Figure 5.5. Each component can be run as its own process on its own machine using AMQP as a message bus. The physical separation helps isolate components that could potentially be compromised.

Let's Encrypt will be running an instance of Boulder. It's performance is limited by our hardware security modules (HSMs). The HSMs are used to sign the certificates and OCSP responses; thus, we have a maximum limit to the number of certificates we can support at any given time. Our HSMs can handle 350 signatures/second. We plan on issuing certificates for 90 days at a time and must refresh our OCSP responses every 3 days, yielding 31 total signatures for every certificate issued. This would yield 87.7 million outstanding certificates, but we expect people to renew their certificates approximately every two months if all the processes are automated. This means that we have a 50% overlap of certificate lifetime, giving us a total capacity of 58.5 million certificates we can service at any given time. Looking at our certificate scanning data, we found 24,442,824 hosts serving HTTPS (the vast majority without trusted certificates) [43]. However, we already have several million expected certificates from web hosting companies months before launch. These companies would like to get certificates for all of their domains that they manage. We expect to drastically increase adoption of HTTPS. Time will tell how long we can stand before we must purchase more hardware. More information about the public's response is available in Section 5.7.2.

5.6.1.1 Security Considerations

Creating an automated CA requires care to avoid misissuance like those made famous over the past few years. [29, 112] There are two primary areas of concern. The first is the actual compromise of the system. In order to help mitigate the risk of compromise and misissuance, Boulder implements a defense-in-depth approach. As a first step, separating and isolating the various components follows the principle of least privilege. Physically separating the components with different security properties and attack surfaces provides an additional layer of defense. For instance, the web front end and validation authority both require direct access to the Internet, and thus have the largest attack surface.

The validation authority has to directly connect to attacker-controlled servers on the Internet, leaving it particularly vulnerable. There are a few techniques that Boulder can employ in order to reduce the chance of compromise. First, the validation authority should not have to save any state. The validation authority can load a safe processing state after each validation. This prevents an attacker from potentially opening a backdoor into the machine in order to gain further access. Another important step is to decrease the attack surface as much as possible. This means to implement the bare minimum in order to meet the functionality necessary for the task. One attractive feature about DVSNI is that the challenge can be retrieved on the initial ServerHello message which contains the certificate. There is no need to finish the complicated TLS handshake or to continue on with any other protocols. The more challenges that are supported, though, does increase the attack surface, so care should be taken when deciding which subset of ACME challenges to implement.

The second area of concern is that the server, components, and protocol are implemented with enough assurance that we properly perform DV and achieve the overarching goals of being more secure and transparent than other CAs. Boulder will support multiple validation authorities to perform multi-path probing techniques for domain validation. Multi-path probing helps prevent man-in-the-middle attacks. The attacker would have to MITM multiple probes from distinct geographic locations. Local MITM attacks would have insufficient

power to convince the CA of domain ownership, not a guarantee provided by all CAs. To aid transparency, Boulder supports Certificate Transparency and will also allow CAs to easily publish the proof used to issue all certificates. Maintaining transparency should increase the public's trust in the CA and also allows third-parties to watch and verify correct behavior. Misissued certificates can be detected before they are used in the wild.

5.6.2 Clients

There are many different models for ACME clients. Basic clients can be created that simply retrieve a certificate, but leaving the installation of the certificate to the user. Clients can also be worked into existing servers. This form of integration would insure that certificates are auto-renewed, but leaves the configuration troubles to the user. Finally, closely integrated outside processes can be used which edit the configurations of applicable servers. For the official Let's Encrypt client, we chose the latter approach for numerous reasons.

In the default case, no configuration is required for the web server at all. All other techniques would require manual setup to get the module or certificates installed. With the outside process approach, a single command can be run. The software can automatically find and enable the necessary modules and work with the existing configuration files. Working within the configuration files also means that we do not increase the attack surface of the server, which is presumably always running. All changes made to the server are transparent; experienced system administrators can analyze the changes and further modify the TLS configuration based on their own needs. The certificates remain accessible and additional tools can be used to manage them. Two of the most basic managerial features are renewal and revocation.

Handling the configuration also allows us to provide guided defaults and easily deploy beneficial TLS and server configuration features. We would like HTTPS to be the default for websites, not just to make it available for users who happen across the secure site or

users of such products as HTTPS Everywhere [50]. Redirection from HTTP to HTTPS can be accomplished easily as an outside process through the configuration, and we can also push technologies like HSTS and OCSP Stapling, which have not been widely deployed. HSTS guarantees that the user will only accept HTTPS for the domain. This prevents many types of MITM attacks. OCSP Stapling has the server return an OCSP response in the initial handshake. This avoids making the client query the CA about the domain. This benefits both the privacy of the end-user and saves the certificate authority valuable bandwidth. Currently, only 4.5% of sites support HSTS and 21.6% support OCSP stapling. [14]

There are also extremely new technologies that can achieve widescale adoption through integration. The W3C working group recently released the "upgrade-insecure-requests" Content Security Policy directive, which blocks mixed content and automatically upgrades all HTTPS requests. It is an easy win for system administrators who want to have secure sites, but it has yet to receive much attention. The Let's Encrypt client can act as a platform for the best security practices, quickly deploying the latest technologies.

Finally, the client's management of configuration files allows for maximal code reuse. Each web server has only to implement an interface to be supported. We have "Authenticator" and "Installer" interfaces. Authenticators are modules that can perform the challenges defined in ACME and Installers are traditionally any server/module that can use a certificate. This separation removes the need for servers that utilize certificates from being able to prove ownership of the domain. Email servers like Exim and Postfix can simply support installation, relying on a separate plugin to handle the authentication. The Let's Encrypt client comes preloaded with a "Standalone" authenticator, which can be used for this very purpose. Thus far, we have developed manual, standalone, Apache, and Nginx authenticators, and we have Apache and Nginx installers. Outside contributors have already developed a DNS based authenticator and an Installer plugin for Icecast, a multimedia streaming server.

Perhaps the primary concern when modifying configuration files is fragility and correctness. The Let's Encrypt client has several built-in mechanisms to help assuage concerns

and provide transparency. First, the client guarantees that backups of the configuration are recorded and saved before any modifications take place. All new configuration files are also catalogued before they are put into place. If at any point the process fails, the configuration can be restored to the original state. In the unlikely event that the program crashes, the client recognizes the inconsistent configuration state at next boot and will revert to the last safe configuration checkpoint. All configuration changes the client makes are displayed in human-readable form and can be rolled back.

The client has also intelligently designed certificate installation into web servers. After the initial installation, the client no longer requires access to the server's configuration files. The certificate files contained in the configurations are symbolic links to the most recent, up-to-date certificate. Likewise, all TLS options that may need to be adapted in the future are contained in a separate file that is linked into the configuration.

Obviously, no matter how much thought and design has gone into a client, it is unlikely to suit everyone's needs. In order to aid in the development, we have separated out all of the protocol code into a separate "ACME" module, making other clients easier to write. GitHub projects have already been developed utilizing the "ACME" module. For instance, websites now often use shared web hosting, where clients do not have full control over the servers running their websites. Hosting providers themselves will have to run an ACME client that works with their existing infrastructure. Several web hosting providers have already expressed an interest in getting certificates for all of the domains that they manage.

5.7 Becoming a CA

There are few different paths to becoming a CA. Buying an existing CA is the fastest path, as the CA is already in root CA programs or trust stores, and the infrastructure and processes have already been implemented. The downsides of buying a CA is that it is a thin market. CAs that cannot buyout an existing CA can apply for inclusion into the trust stores themselves. The drawback of this approach is that it requires several months of preparation

and the CA will not be accepted by older clients who do not update their trust store. As a commercial or general-purpose CA, this is not acceptable. Finally, you can become an intermediate CA for an existing CA. The intermediate CA is trusted by all clients that trust the root immediately, but you are limited to the signing policies and practices of the root certificate authority. There is also an associated risk that if the root CA gets revoked for any reason, the intermediate CA is revoked too.

Given these tradeoffs, Let's Encrypt has applied to the root programs as a new CA. Additionally though, Let's Encrypt has secured a cross-signature by a currently trusted root CA, IdenTrust. A cross-signature implies their root certificate will sign the Let's Encrypt intermediate CA public key. This allows the Let's Encrypt Intermediate CA to chain to IdenTrust, while the Let's Encrypt Root CA propagates. Let's Encrypt will be able to operate independently of existing CAs, while still having day one compatibility with all clients.

There are three main root CA programs, Mozilla, Microsoft, and Apple. Linux and Firefox users use Mozilla's root store, while users of Google's Chrome browser trust the underlying OS's root store. Although each store is separate, they all generally have the same requirements. CAs generally apply to be included in all three. As we saw in Chapter II, 99.4% of server certificates are signed by CAs trusted by all three CA stores.

In order to be included in the trust stores, or be an approved intermediate CA, you must first pass an onerous standardized audit. The two widely-accepted audits are the WebTrust Principles and Criteria for Certification Authorities and ETSI TS 102. The WebTrust audit, which Let's Encrypt performed, contains around 50 pages of checklist criteria ranging from personnel security to CA Key Compromise. The audit also verifies the governing documents of the CA, the Certificate Policy (CP) and Certificate Practice Statement (CPS).

The CP is designed to delineate the various components of the PKI, their roles and responsibilities. The CPS's aim is to describe the practices of the certificate authority. It describes all processes and standards the CA must follow. Combined, Let's Encrypt's CP and CPS are nearly 200 pages. Both the CP and CPS must follow all of the guidelines set forth

in the 94-page RFC 3647 [34]. Describing, documenting, and verifying these documents and practices requires tremendous effort from a collaborative team of technologists and lawyers. It is expensive for the CA, both from a personnel and financial standpoint.

Perhaps the most interesting requirements are those of the CA key generation, which must be performed in a ceremony. They are witnessed by an independent party and videotaped; everything must be scripted and logged. Everyone is prescribed a clearly-defined role beforehand. The hardware preparation, operating system installation, CA installation/configuration, key backup, signing and shutdown must be prepared and documented beforehand. The ceremony must meet physical security requirements, all materials must be stored according to plan and tested appropriately. Finally, the CA keys must be created and stored in approved ISO 15782-1/FIPS 140-2 hardware security modules. Any deviation from the procedures or errors in the script result in the rescheduling and restarting of the ceremony.

The WebTrust audit must be completed every year. In addition, CAs without a currently valid audit are required to perform a point-in-time readiness assessment. Essentially, the CA must be monitored for a month before they can begin issuing publicly-trusted certificates. Ironically, part of the audit is to search the certificates our team at the University of Michigan provides, at `scans.io`, in order to check for inconsistencies. After the point-in-time readiness assessment has been completed, a normal full WebTrust audit must be completed within 90 days of the first publicly-trusted certificate.

CAs must also now comply with the CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates [32] which is produced by the CABForum. The CABForum is a conglomeration of CAs and browsers that discuss policies and best practices with which both parties agree to abide by. As expected, the WebTrust audit is heavily influenced by the CABForum Baseline Requirements.

Although there is an extreme amount of documentation and audits required by CAs, it does not mean that certificate authorities do not violate their own policies. Fortunately, as

transparency technologies like our collection of certificates at scans.io [43] and certificate transparency [83] have become standard, the gap between policy and practice has been decreasing [37].

5.7.1 Industry Response

Let's Encrypt and the ACME protocol is a very disruptive technology. Current individual certificate authorities make tens of millions of dollars each year selling domain-validated certificates. Certainly, the appearance of a free CA that is easier to use can drastically affect existing CAs' bottom lines. In addition, nation-states that have a vested interest in censorship cannot be sanguine with HTTPS becoming cheaper and ubiquitous.

Surprisingly, the response has been largely positive. DigiCert, a company which sells OV certificates starting at $139, has denounced free certificates (DV) as not providing enough security [91] or the same security as their premium OV certificates. Most articles and view points have been positive regarding the initiative. In fact, a few additional CAs and hosting providers have decided to offer free DV certificates in the wake of the Let's Encrypt announcement. WoSign, a Chinese certificate authority, began issuing free certificates valid for 2 years, and EuroDNS began offering free certificates to their customers in February and March of 2015 respectively. [7] [3] Additionally, CAs have started to compete from the usability perspective, introducing new certificate installation tools of their own after the announcement of Let's Encrypt. [90] Let's Encrypt is demanding change in the industry, and it is fulfilling to see the ideals and core mission of Let's Encrypt being adopted.

5.7.2 Consumer Response

The consumer response has been overwhelmingly positive, as one might expect from the announcement of a free product. The Let's Encrypt demo page has space for comments and feedback. Having been viewed nearly 50,000 times at writing, it received 384 positive and 5 negative remarks. There are 39 top-level comments, 26 of which were positive, 1 was

negative, and the rest were neutral. The neutral comments either asked about the capabilities of the system or were simply used to share the demo with their friends and colleagues.

The concerns are of the following general form.

- How can Let's Encrypt possibly be secure if everything is automated?
- Let's Encrypt puts the Internet at risk, being located in the United States and subject to the laws of the United States government.
- Free certificates are bad for the Internet because it enables malware to encrypt their traffic.

The first two are a result of customers not understanding how certificate authorities currently operate and the trust model of the system in general. To reiterate, we are performing the same or strictly stronger domain validation checks that other CAs are currently performing. We also have the additional security afforded by the checks described throughout the ACME and Boulder sections.

The second concern does not consider that the United States government already has access to their own CAs and under the court of law, over 30% of the world's CAs. [43] In fact, Let's Encrypt is unlikely to be targeted by governments for a variety of reasons.

First, Let's Encrypt has been designed to be as transparent as possible. Let's Encrypt supports certificate transparency and is designed to allow us to publish records of every authorization. The ISRG, the non-profit running Let's Encrypt, has already published its first legal transparency report, months before the service has even begun to issue certificates. [69] The ISRG has already announced the expected schedule of transparency reports which acts as a warrant canary. If the report is ever withheld or late, its subscribers know the government intervened. Additionally, by reporting so early, the ISRG was able to choose the format of the report and actually report "0" under each legal order. Twitter and other companies have been forced by the government to report ranges of numbers which include

"0". Publishing the legal transparency report before the chance for it to be subverted is an important first step to maintaining trust and gaining user confidence.

In this regard, the ISRG also benefits from being sponsored by the Electronic Frontier Foundation (EFF). The EFF has a team of lawyers with a long history focused on protecting user's digital rights.

The final concern, that we are enabling malware to more easily encrypt their traffic with browser-trusted certificates, is factual. Malware often does encrypt their traffic, but commonly uses self-signed certificates or their own PKI. Anti-virus companies rely on these trademarks to identify the malware. Now that browser-trusted certificates can be attained automatically, free of charge, the malware can potentially be able to blend in more easily with benign traffic. Let's Encrypt will launch with support for Google's safe-browsing API, which should reduce malicious phishing attempts and make it more difficult to get certificates for malware.

Any malware or phishing detection system *cannot* be perfect, though. At any point in time, a benign website may be altered to include "malicious" or "phishing" content, whether under the domain owner's own volition or involuntarily due to compromise. Although this can be construed as a negative side-effect, determined malware authors already do attain very cheap server certificates for their software [48]. CAs cannot be in the business to police content on the Internet.

Arguments against free automated certificates because of the potential to increase malicious and fraudulent activity are analogous to the arguments in the ongoing Crypto Wars. We believe the benefit of enabling encryption and secure sessions for all greatly outweigh any perceived drawbacks.

5.8 Conclusion

HTTPS deployment has lagged severely behind the adoption of HTTP. Two of the main factors hindering the deployment of HTTPS have been the cost of certificates and the time

necessary to deploy HTTPS correctly. Current certificate authorities have taken advantage of their market situation and consumer naivety, hindering the security and privacy of the Internet. Our analysis of current CA practices revealed that the marginal cost of issuing DV certificates is approximately zero. The whole DV process can be automated while also increasing the security of the CAs and ecosystem at large. We developed ACME, a protocol that handles all the functionality of a traditional certificate authority, allowing the automatic adoption of TLS into end-products. ACME fills a void that has been missing from production systems and, subsequently, an ACME IETF working group has been formed. Given our ACME protocol, we have founded Let's Encrypt, a completely free and automated certificate authority that has the potential to drastically alter the deployment of TLS. We have developed implementations of both a CA and extensible client which have been released open source. The technology has already affected current CAs, improving the ecosystem. Let's Encrypt itself has received positive reception; more than 18,000 individuals have already signed up for the beta program.

CHAPTER VI

Conclusion and Future Work

This thesis has demonstrated how measurement-based security and automation can reduce the vulnerabilities originating from both certificate authority practice and HTTPS server deployments. In Chapter II we performed the first systematic and longitudinal study of the HTTPS ecosystem and in Chapter III we analyzed how the ecosystem responds to upheaval. Our analysis revealed a number of worrying trends, but we focused on two of the larger issues. First, we discovered the PKI has an extremely large attack surface. An attacker must only compromise a single CA to defeat the guarantees provided by HTTPS and we found 683 different organizations with certificate signing power. The second issue we focused on was the exorbitant cost of HTTPS. We discovered many symptoms of the high cost of HTTPS, including, low adoption rates, a large percentage of misconfigured servers, and the observed failure of system administrators to maintain the security of their HTTPS deployments.

In Chapter IV we presented CAge, a mechanism which can be applied to certificate authorities to reduce the attack surface of HTTPS. CAge infers CAs' signing behavior and develops signing rules based on the inferences. Even in its most basic configuration, the attack surface of CAs can be reduced by 90%. In Chapter V we presented Let's Encrypt, the first completely automated certificate authority. Let's Encrypt aims to drastically increase adoption of HTTPS by offering both free certificates and making it significantly easier to

deploy HTTPS through the use of automated clients. We developed a new protocol, ACME, which automates all CA operations and we developed new validation methods to increase the security of PKI in general. Finally, we described the process of building a CA and the insights gleaned from its development.

Although this thesis has provided a path to a significantly stronger PKI on the Internet, there are some problems we identified through our analysis which warrant further investigation and can likely be solved through further additions to the ACME protocol.

6.1 Future ACME Extensions

Let's Encrypt should stand as an exemplar and raise the bar for CA security. ACME's simplicity and extensibility can allow it to solve other problems posed with PKI.

6.1.1 PKI Trust Agility

Marlinspike introduced the concept of trust agility [88] which implies that you can choose who you trust and you are free to change who you trust at any time. The current certificate authority system's trust is, unfortunately, extremely rigid. Individual users have no control over which CAs they trust, and many CAs' trust can never be revoked. Large CAs have deployed certificates on hundreds of thousands of servers that would all become invalid if the CA's trust is revoked from browsers. This would throw spurious errors to users and would break many applications. Indeed, many CAs have been granted leniency, even when they repeatedly demonstrate less than adequate security practices.

In 2011, Comodo was hacked and subsequently misissued certificates for several high-profile sites [112]. This was not their first security failure [135]. Browsers could only respond by blacklisting the particular certificates Comodo claimed they misissued. A much preferred approach would have been to eliminate all certificates from the CA that was compromised. There may have been other certificates created by the attacker that evaded Comodo's detection. In 2015, CNNIC was found issuing an intermediary certificate authority

to an outside company to MITM their customers [81]. The violation caused great debate among the browsers. Ultimately, both Google and Mozilla decided to remove the CNNIC root CA from their browsers, but add in additional code that white-lists certificates from CNNIC signed before the date of removal. This additional code per compromise does not scale well and introduces unnecessary complications into browsers.

6.1.1.1 Directory Servers

ACME directory servers have the potential to alleviate some of the trust agility problems of the current ecosystem. An ACME directory server is a server that maintains a list of active ACME CAs and their associated directory URLs. Clients can be configured with a stable URL that will always direct them to the latest and maintained ACME CAs, which assists with the problem of CA discovery.

Optimally, directory servers would also include metadata about the different CAs and their current offerings. The metadata would be restricted to facts and avoid security theater, which is rampant in the market today. Clients could then choose among the offerings, fostering a free, open, and competitive market for certificates.

Directory servers also have the potential to alleviate some of the trust agility problems. If a CA ever warrants removal from the list of trusted CAs, they could be removed from the directory server. Clients renewing their certificates would choose other CAs, thus depleting the server base. When the directory server is combined with the technique of short-lived certificates, the whole Internet could be transitioned to new, secure CAs and the failed CA could be removed from browsers entirely. This would hold CAs accountable and provides a degree of trust agility for the Internet.

6.1.2 Extending ACME Challenges and Identifiers

ACME represents a general framework to distribute and verify public keys. The challenge framework allows for future communication mediums to be easily supported. Appropriate

challenges need to be developed to support the medium, but the challenges can be directly plugged into the protocol, much like new cipher suites are added to TLS.

There has already been interest in extending ACME to support ephemeral identity certificates for email. The initial idea is to use OAUTH 2.0 to verify email addresses. Requesters would be redirected to an OAUTH server for a token, which would be returned to the CA. The CA would verify the token and thus verify the email address.

Another area that has taken interest in ACME is future Internet architectures. Name Data Networking (NDN) is a future-internet architecture where every piece of data is cryptographically signed [133]. However, in order to verify signatures accurately and take appropriate measures, the classic PKI problems of key management and distribution must be overcome. Given the potential adoption and scale of key management for NDN, simplicity and low-cost is of primary importance.

ACME can easily be applied for names within NDN. The main challenge with applying ACME to NDN is developing a small set of challenges that convey the appropriate authority over identifiers within NDN. Luckily, NDN is ripe with opportunities. Challenges have already been proposed involving NDN scoped-interests and bearer tokens. Additional work will need to be completed in order to develop and consider NDN's unique security parameters.

6.2 Further Analysis

Let's Encrypt launches to the general public in mid-November and its affect on the certificate ecosystem will be realized. As a first order, we will want to determine whether we have succeeded in our goals of increasing the adoption of HTTPS, decreasing server configuration errors, and decreasing the response time of fixing security-critical vulnerabilities.

We can continue the evaluations we performed in chapters II and III. In essence, the same metrics that we utilized to deduce the issues with HTTPS deployment can also determine how effective the launch of Let's Encrypt was in solving them.

This technique does not grant the granularity required for a full evaluation, though. There are many facets to Let's Encrypt, and analyzing each part is important in order to understand the system as a whole. In particular, we should analyze individual implementations and evaluate how well they each achieve their goals.

The client's main goals are usability and the increased adoption of best security practices. Both of these can be roughly measured through more extensive scans. Preparing for this analysis, we have equipped the official client with a user-agent string which should allow us to track the effectiveness of our client compared to the rest of the habits of the Internet as well as Let's Encrypt clients developed by third-parties. Usability can be tracked by the relative adoption of the client and by analyzing the audit logs of Boulder, which provide data about how the client is executed in practice. The client's success at championing best security practices can be measured by the impact of the client on the rates of adoption of the supported security and privacy enhancements. The client aims to make all of the following significantly easier: enabling HSTS, enabling OCSP Stapling, redirecting from HTTP to HTTPS, and adding Content Security Policy directives.

One downside of relying on network monitoring and scanning is that it only grants correlations. Another possible method for evaluating the efficacy of the Let's Encrypt client would be to conduct usability studies. Usability studies would give us greater detail into how system administrators use the tool and would let us know, directly, which features succeeded and which features need to be improved upon.

Boulder, the Let's Encrypt CA implementation, has a different set of goals and will require a different analysis. Boulder primarily aims to be secure and performant. In order to get data regarding these goals, we must analyze the logs and performance of the CA.

Security rests on the number of successful attacks against the CA. In particular, we are interested in attacks that cause misissuance. We should collect data regarding both the types and frequency of attacks against Let's Encrypt. It is likely that certain validation methods are weaker in practice than others. By collecting this data, we can determine the relative

costs of supporting each challenge type. This data is desperately needed by the CA/Browser Forum Validation Group as they work to standardize domain validation techniques [15].

Attacks on Boulder can also help form the development of future challenges and CA requirements. We can determine which attack types are most effective and research ways to help mitigate the attacks. For instance, in the current ACME specification, all of the domain validation techniques are subject to attacks on BGP. Analyzing Let's Encrypt will help us learn the frequency of exploits and the value of additional stop-gap challenges, like the defined proof-of-possession challenge in ACME.

Boulder's performance goal is important as we would like to support as much of the Internet as possible. Although our preliminary tests have shown that our bottleneck is our HSMs, the launch of Let's Encrypt will enable us to see how Boulder scales under real-world load and deployment. Analyzing Boulder for performance issues and bottlenecks will likely yield ways to further improvements and enable us to offer HTTPS to an ever-increasing set of users.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Alexa Top 1,000,000 Sites. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[2] Comodo Relying Party Warranty. https://www.comodo.com/repository/docs/SSL_relying_party_warranty.php.

[3] EuroDNS introduces free SSL certificates to customers. http://news.eurodns.com/eurodns-introduces-free-ssl-certificates-to-customers/.

[4] Let's Encrypt. https://letsencrypt.org.

[5] Tracking the FREAK attack. https://freakattack.com.

[6] What is shellshock? https://shellshocker.net/.

[7] WoSign: Free 2y multi-domain ssl certificate. https://www.ohling.org/blog/2015/02/wosign-free-2y-ssl-certificate.html.

[8] Gmail.com SSL MITM Attack by Iranian government, Aug. 2011. http://pastebin.com/ff7Yg663.

[9] Bug 581901 - Add HARICA root certificate. Website, Apr. 2012. https://bugzilla.mozilla.org/show_bug.cgi?id=581901.

[10] Facts about ipsCA, Apr. 2013. http://certs.ipsca.com/companyIPSipsCA/competitorssay.asp.

[11] WebTrust for Certification Authorities — SSL Baseline Requirements Audit Criteria v.1.1, Jan. 2013. http://www.webtrust.org/homepage-documents/item72056.pdf.

[12] Guidelines For The Issuance And Management Of Extended Validation Certificates, Oct. 2014. https://cabforum.org/wp-content/uploads/EV-V1_5_2Libre.pdf.

[13] Heartbleed F.A.Q., 2014. https://www.startssl.com/?app=43.

[14] SSL Pulse, Apr. 2014. https://www.trustworthyinternet.org/ssl-pulse/.

[15] CA/Browser Forum Validation Working Group, 2015. https://cabforum.org/current-work/validation-working-group/.

[16] GoDaddy, Oct. 2015. https://www.godaddy.com/web-security/ssl-certificate.

[17] ADRIAN, D., BHARGAVAN, K., DURUMERIC, Z., GAUDRY, P., GREEN, M., HALDERMAN, J. A., HENINGER, N., SPRINGALL, D., THOMÉ, E., VALENTA, L., VANDERSLOOT, B., WUSTROW, E., ZANELLA-BÉGUELIN, S., AND ZIMMERMANN, P. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 5–17.

[18] AKHAWE, D., AMANN, B., VALLENTIN, M., AND SOMMER, R. Here's my cert, so trust me, maybe? Understanding TLS errors on the web. In *Proceedings of the 22nd international conference on the World Wide Web* (2013), pp. 59–70.

[19] AKHAWE, D., AND FELT, A. P. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Usenix Security* (2013), pp. 257–272.

[20] AL FARDAN, N., AND PATERSON, K. Lucky Thirteen: Breaking the TLS and DTLS record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on* (May 2013), pp. 526–540.

[21] ALICHERRY, M., AND KEROMYTIS, A. D. Doublecheck: Multi-path verification against man-in-the-middle attacks. In *ISCC* (2009), IEEE, pp. 557–563.

[22] AMANN, B., VALLENTIN, M., HALL, S., AND SOMMER, R. Extracting Certificates from Live Traffic: A Near Real-Time SSL Notary Service. Tech. Rep. TR-12-014, ICSI, Nov. 2012.

[23] ASGHARI, H., VAN EETEN, M., ARNBAK, A., AND VAN EIJK, N. Security economics in the HTTPS value chain. *Available at SSRN 2277806* (2013).

[24] BARKER, E., BARKER, W., BURR, W., POLK, W., AND SMID, M. Recommendation for Key Management - Part 1: General (Revision 3), July 2012. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf.

[25] BARKER, E., AND ROGINSKY, A. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths, Jan. 2011. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf.

[26] BARNES, R. Deprecating non-secure HTTP, Apr. 2015. https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/.

[27] BARNES, R., HOFFMAN-ANDREWS, J., AND KASTEN, J. Automatic Certificate Management Environment (ACME), July 2015. https://tools.ietf.org/html/draft-barnes-acme-04.

[28] BELLO, L. DSA-1571-1 OpenSSL—Predictable random number generator, 2008. Debian Security Advisory. http://www.debian.org/security/2008/dsa-1571.

[29] BHAT, S. Gmail users in Iran hit by MITM attacks, Aug. 2011. http://techie-buzz.com/tech-news/gmail-iran-hit-mitm.html.

[30] BLAKE-WILSON, S., NYSTROM, M., HOPWOOD, D., MIKKELSEN, J., AND

WRIGHT, T. Transport Layer Security (TLS) Extensions. RFC 3546 (Proposed Standard), June 2003.

[31] CA/BROWSER FORUM. Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.1, Nov. 2011. https://www.cabforum.org/Baseline_Requirements_V1_1.pdf.

[32] CA/BROWSER FORUM. Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.3, Apr. 2015. https://cabforum.org/wp-content/uploads/CAB-Forum-BR-1.3.0.pdf.

[33] CAMP, L., AND JOHNSON, M. Defeating the greatest masquerade. In *The Economics of Financial and Medical Identity Theft*. Springer US, 2012, pp. 31–39.

[34] CHOKHANI, S., FORD, W., SABETT, R., MERRILL, C., AND WU, S. Internet X.509 public key infrastructure certificate policy and certification practices framework. https://tools.ietf.org/html/rfc3647.

[35] COMODO GROUP, INC. Alternative methods of domain control validation (DCV). https://support.comodo.com/index.php?/Default/Knowledgebase/Article/View/791/16/alternative-methods-of-domain-control-validation-dcv.

[36] COOPER, D., SANTESSON, S., FARRELL, S., BOEYEN, S., HOUSLEY, R., AND POLK, W. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[37] DELIGNAT-LAVAUD, A., ABADI, M., BIRRELL, A., MIRONOV, I., WOBBER, T., AND XIE, Y. Web PKI: Closing the gap between guidelines and practices. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium* (2014).

[38] DHAMIJA, R., AND TYGAR, J. Phish and hips: Human interactive proofs to detect phishing attacks. In *Human Interactive Proofs*, H. Baird and D. Lopresti, Eds., vol. 3517 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, pp. 127–141.

[39] DIERKS, T., AND RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[40] DURUMERIC, Z., ADRIAN, D., KASTEN, J., SPRINGALL, D., BAILEY, M., AND HALDERMAN, J. A. POODLE Attack and SSLv3 Deployment. https://poodle.io/.

[41] DURUMERIC, Z., ADRIAN, D., MIRIAN, A., KASTEN, J., BURSZTEIN, E., LIDZBORSKI, N., THOMAS, K., ERANTI, V., BAILEY, M., AND HALDERMAN, J. A. Neither snow nor rain nor MITM... an empirical analysis of email delivery security. In *Proceedings of the Internet Measurement Conference* (2015).

[42] DURUMERIC, Z., KASTEN, J., ADRIAN, D., HALDERMAN, J. A., BAILEY, M., LI, F., WEAVER, N., AMANN, J., BEEKMAN, J., PAYER, M., AND PAXSON, V. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (New York, NY, USA, 2014), IMC '14, ACM, pp. 475–488.

[43] DURUMERIC, Z., KASTEN, J., BAILEY, M., AND HALDERMAN, J. A. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 13th Internet Measurement Conference* (Oct. 2013).

[44] DURUMERIC, Z., WUSTROW, E., AND HALDERMAN, J. A. ZMap: Fast Internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium* (Aug. 2013).

[45] EASTLAKE, D. Transport Layer Security (TLS) extensions: Extension definitions. https://tools.ietf.org/html/rfc6066.

[46] ECKERSLEY, P. Sovereign Key Cryptography for Internet Domains. Website, Nov. 2011. https://www.eff.org/sovereign-keys.

[47] ECKERSLEY, P., AND BURNS, J. An observatory for the SSLiverse. Talk at Defcon 18 (2010). https://www.eff.org/files/DefconSSLiverse.pdf.

[48] EDGECOMBE, G. Certificate authorities issue SSL certificates to fraudsters. http://news.netcraft.com/archives/2015/10/12/certificate-authorities-issue-hundreds-of-deceptive-ssl-certificates-to-fraudsters.html.

[49] ELECTRONIC FRONTIER FOUNDATION. The EFF SSL observatory. https://www.eff.org/observatory.

[50] ELECTRONIC FRONTIER FOUNDATION. HTTPS Everywhere.

[51] ELLIS, A. Akamai heartbleed update (v3), Apr. 2014. https://blogs.akamai.com/2014/04/heartbleed-update-v3.html.

[52] EVANS, C. New Chromium security features, June 2011. Website, 2011. http://blog.chromium.org/2011/06/new-chromium-security-features-june.html.

[53] FIELDING, R. Fielding dissertation: Chapter 5: Representational state transfer (REST), 2000.

[54] FOX IT. Deep dive into QUANTUM INSERT, Apr. 2015. http://blog.fox-it.com/2015/04/20/deep-dive-into-quantum-insert/.

[55] GAL, A. Data is at the heart of search, but who has access to it?, Feb. 2015. http://andreasgal.com/2015/03/30/data-is-at-the-heart-of-search-but-who-has-access-to-it/.

[56] GODADDY. Verifying your domain ownership for SSL certificate requests (HTML or DNS). https://www.godaddy.com/help/verifying-your-domain-ownership-for-ssl-certificate-requests-html-or-dns-7452.

[57] GOODIN, D. DDoS attacks that crippled GitHub linked to Great Firewall of China, Apr. 2015. http://arstechnica.com/security/2015/04/ddos-attacks-that-crippled-github-linked-to-great-firewall-of-china/.

[58] GRANT, S. The Bleeding Hearts Club: Heartbleed Recovery for System Administrators, Apr. 2014. https://www.eff.org/deeplinks/2014/04/bleeding-hearts-club-heartbleed-recovery-system-administrators.

[59] GREENWALD, G., AND MACASKILL, E. NSA Prism program taps into user data of Apple, Google and others, June 2013.

[60] GREGORIO, J., FIELDING, R., HADLEY, M., NOTTINGHAM, M., AND ORCHARD, D. URI template. https://tools.ietf.org/html/rfc6570.

[61] GRUBB, B. Heartbleed Disclosure Timeline: Who Knew What and When. http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html.

[62] HENINGER, N. Factoring as a service, Aug. 2013. Talk at CRYPTO Rump Session 2013. http://crypto.2013.rump.cr.yp.to/981774ce07e51813fd4466612a78601b.pdf.

[63] HENINGER, N., DURUMERIC, Z., WUSTROW, E., AND HALDERMAN, J. A. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium* (Aug. 2012).

[64] HOLZ, R., BRAUN, L., KAMMENHUBER, N., AND CARLE, G. The SSL landscape: A thorough analysis of the X.509 PKI using active and passive measurements. In *11th ACM SIGCOMM conference on Internet measurement (IMC)* (2011).

[65] HOUSLEY, R., FORD, W., POLK, W., AND SOLO, D. Internet X.509 public key infrastructure certificate and CRL profile.

[66] IANA. Iana top level domains. Website. http://data.iana.org/TLD/tlds-alpha-by-domain.txt.

[67] INCOMMON. InCommon certificate manager - initiating domain control validation (DCV). https://www.incommon.org/certificates/repository/Initiating_DCV_in_InCommon_CM.pdf.

[68] INTERNET SECURITY RESEARCH GROUP. Let's Encrypt Repository. https://github.com/letsencrypt.

[69] INTERNET SECURITY RESEARCH GROUP. Legal transparency report, July 2015. https://letsencrypt.org/documents/ISRG-Legal-Transparency-Report-July-1-2015.pdf.

[70] JACKSON, C., SIMON, D. R., TAN, D. S., AND BARTH, A. An evaluation of extended validation and picture-in-picture phishing attacks. In *Proceedings of the 11th International Conference on Financial Cryptography and 1st International Conference on Usable Security* (Berlin, Heidelberg, 2007), FC'07/USEC'07, Springer-Verlag, pp. 281–293.

[71] JONES, M. JSON Web Key (JWK). https://tools.ietf.org/html/rfc7517.

[72] JONES, M., BRADLEY, J., AND SAKIMURA, N. JSON Web Signature (JWS). Internet Requests for Comments, May 2015. https://tools.ietf.org/html/rfc7515.

[73] KAMINSKY, D. Black Ops 2008: Its The End Of The Cache As We Know It, Aug. 2008. BlackHat USA 2008.

[74] KASTEN, J., WUSTROW, E., AND HALDERMAN, J. A. Cage: Taming certificate authorities by inferring restricted scopes. In *17th International Conference on Financial Cryptography and Data Security (FC)* (2013).

[75] KLEINJUNG, T., AOKI, K., FRANKE, J., LENSTRA, A., THOMÉ, E., BOS, J., GAUDRY, P., KRUPPA, A., MONTGOMERY, P., OSVIK, D., ET AL. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 333–350.

[76] KLYNE, G., AND NEWMAN, C. Date and time on the Internet: Timestamps. https://tools.ietf.org/html/rfc3339.

[77] KRAVETS, D. Comcast Wi-Fi serving self-promotional ads via JavaScript injection, Sept. 2014. http://arstechnica.com/tech-policy/2014/09/why-comcasts-javascript-ad-injections-threaten-security-net-neutrality/.

[78] LANGLEY, A. Enhancing digital certificate security. Google Online Security Blog, http://googleonlinesecurity.blogspot.com/2013/01/enhancing-digital-certificate-security.html, Jan. 2013.

[79] LANGLEY, A. No, Don't Enable Revocation Checking, Apr. 2014. https://www.imperialviolet.org/2014/04/19/revchecking.html.

[80] LANGLEY, A. Revocation Still Doesn't Work, Apr. 2014. https://www.imperialviolet.org/2014/04/29/revocationagain.html.

[81] LANGLEY, A. Maintaining digital certificate security. Google Online Security Blog, http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html, Mar. 2015.

[82] LAURIE, B., LANGLEY, A., AND KASPER, E. Certificate transparency. Website, Sept. 2012. http://tools.ietf.org/html/draft-laurie-pki-sunlight-00.

[83] LAURIE, B., LANGLEY, A., AND KASPER, E. Certificate transparency.

[84] LENSTRA, A. K., HUGHES, J. P., AUGIER, M., BOS, J. W., KLEINJUNG, T., AND

WACHTER, C. Ron was wrong, Whit is right. *IACR Cryptology ePrint Archive 2012* (2012), 64.

[85] LOESCH, C. Certificate patrol. Website. http://patrol.psyced.org/.

[86] MARCZAK, B., WEAVER, N., DALEK, J., ENSAFI, R., FIFIELD, D., MCKUNE, S., REY, A., SCOTT-RAILTON, J., DEIBERT, R., AND PAXSON, V. China's great cannon, Apr. 2015. https://citizenlab.org/2015/04/chinas-great-cannon/.

[87] MARLINSPIKE, M. Defeating OCSP With The Character '3', July 2009. http://www.thoughtcrime.org/papers/ocsp-attack.pdf.

[88] MARLINSPIKE, M. SSL and the Future of Authenticity, Aug. 2011. BlackHat USA 2011.

[89] MARQUESS, S. Of Money, Responsibility, and Pride, Apr. 2014. http://veridicalsystems.com/blog/of-money-responsibility-and-pride/.

[90] MARTINS, F. Certificate monitoring and express install SSL management made easy, Apr. 2015. https://blog.digicert.com/certificate-monitoring-express-install/.

[91] MARTINS, F. The fraud problem with free SSL certificates, Apr. 2015. https://blog.digicert.com/fraud-problem-with-free-ssl-certificates/.

[92] MASNICK, M. Shameful Security: StartCom Charges People To Revoke SSL Certs Vulnerable to Heartbleed, Apr. 2014. http://www.techdirt.com/articles/20140409/11442426859/shameful-security-startcom-charges-people-to-revoke-ssl-certs-vulnerable-to-heartbleed.shtml.

[93] MATHEWSON, N., AND PROVOS, N. libevent—An event notification library. http://libevent.org.

[94] MAYER, J. The turn-verizon zombie cookie, Jan. 2015. http://webpolicy.org/2015/01/14/turn-verizon-zombie-cookie/.

[95] MCMILLAN, R. Verizons Perma-Cookie Is a Privacy-Killing Machine, Oct. 2014. http://www.wired.com/2014/10/verizons-perma-cookie/.

[96] MEHTA, N., AND CODENOMICON. The heartbleed bug. http://heartbleed.com/.

[97] MOZILLA. Mozilla CA certificate policy (version 2.1). Website, Feb. 2013. https://www.mozilla.org/projects/security/certs/policy/.

[98] NCCGROUP. Details on the CRIME attack, Sept. 2012. https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2012/september/details-on-the-crime-attack/.

[99] NETCRAFT, LTD. Web server survey. http://news.netcraft.com/archives/2013/05/03/may-2013-web-server-survey.html, May 2013.

[100] NETMARKETSHARE. Desktop operating system market share, Apr. 2013. http://www.netmarketshare.com/operating-system-market-share.aspx.

[101] NETWORK, M. D. Mozilla network security services (nss). http://www.mozilla.org/projects/security/pki/nss/.

[102] NIGHTINGALE, J. Comodo Certificate Issue – Follow Up, Mar. 2011. https://blog.mozilla.org/security/2011/03/25/comodo-certificate-issue-follow-up/.

[103] NOTTINGHAM, M. Web linking. Internet Requests for Comments, Oct. 2010. https://tools.ietf.org/html/rfc5988.

[104] NOTTINGHAM, M., AND WILDE, E. Problem details for HTTP APIs.

[105] NYSTROM, M., AND KALISKI, B. PKCS #10: Certification request syntax specification version 1.7. https://tools.ietf.org/html/rfc2986.

[106] OPENSSL PROJECT TEAM. OpenSSL Security Advisory, Apr. 2014. http://www.mail-archive.com/openssl-users@openssl.org/msg73408.html.

[107] OPENSSL PROJECT TEAM. OpenSSL Version 1.0.1g Released, Apr. 2014. http://www.mail-archive.com/openssl-users@openssl.org/msg73407.html.

[108] OWASP. Session hijacking attack. https://www.owasp.org/index.php/Session_hijacking_attack.

[109] PINCKAERS, W. http://lekkertech.net/akamai.txt.

[110] PRINCE, M. The Hidden Costs of Heartbleed, Apr. 2014. http://blog.cloudflare.com/the-hard-costs-of-heartbleed.

[111] RESCORLA, E. *SSL and TLS: designing and building secure systems*, vol. 1. Addison-Wesley Reading, 2001.

[112] RICHMOND, R. An Attack Sheds Light on Internet Security Holes. *The New York Times* (Apr. 2011). http://www.nytimes.com/2011/04/07/technology/07hack.html.

[113] RISTIC, I. Internet SSL survey 2010. Talk at BlackHat 2010. http://media.blackhat.com/bh-ad-10/Ristic/BlackHat-AD-2010-Ristic-Qualys-SSL-Survey-HTTP-Rating-Guide-slides.pdf.

[114] SANDVINE. Global Internet Phenomena Spotlight: Encrypted Internet Traffic, Apr. 2015. https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/encrypted-internet-traffic.pdf.

[115] SCHNEIER, B. Man-in-the-middle attack against SSL 3.0/TLS 1.0, Sept. 2011. https://www.schneier.com/blog/archives/2011/09/man-in-the-midd_4.html.

[116] SCHNEIER, B. How the NSA attacks Tor/Firefox users with QUANTUM and FOXACID, Oct. 2013. https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html.

[117] SEGGELMANN, R., TUEXEN, M., AND WILLIAMS, M. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. IETF Request for Comments (RFC) 6520, February 2012. https://tools.ietf.org/html/rfc6520.

[118] SLEEVI, R., EVANS, C., AND PALMER, C. Public key pinning extension for HTTP. https://tools.ietf.org/html/rfc7469.

[119] SOGHOIAN, C., AND STAMM, S. Certified lies: detecting and defeating government interception attacks against ssl (short paper). In *Proceedings of the 15th international conference on Financial Cryptography and Data Security* (Berlin, Heidelberg, 2012), FC'11, Springer-Verlag, pp. 250–259.

[120] SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A., MOLNAR, D., OSVIK, D. A., AND DE WEGER, B. MD5 considered harmful today. http://www.win.tue.nl/hashclash/rogue-ca/.

[121] STARTCOM. StartSSL comparison chart, Oct. 2015. https://startssl.com/?app=40.

[122] SULLIVAN, N. The Results of the CloudFlare Challenge. http://blog.cloudflare.com/the-results-of-the-cloudflare-challenge.

[123] SUNSHINE, J., EGELMAN, S., ALMUHIMEDI, H., ATRI, N., AND CRANOR, L. F. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the 18th conference on USENIX security symposium* (Berkeley, CA, USA, 2009), SSYM'09, USENIX Association, pp. 399–416.

[124] SYMANTEC. SSL certificates, Oct. 2015. https://www.symantec.com/ssl-certificates/.

[125] TENNISON, J. Good practices for capability URLs. *FPWD capability-urls, February* (2014).

[126] TRUSTWAVE. Validation procedures, Oct. 2015. https://ssl.trustwave.com/support/support-validation.php.

[127] VIEGA, J., MESSIER, M., AND CHANDRA, P. *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly, 2002.

[128] VIXIE, P., AND DAGON, D. Use of bit 0x20 in DNS labels to improve transaction identity. *Work in progress, Internet Engineering Task Force* (2008).

[129] WENDLANDT, D., ANDERSEN, D. G., AND PERRIG, A. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In *USENIX 2008 Annual Technical Conference* (Berkeley, CA, USA, 2008), USENIX Association, pp. 321–334.

[130] WHALEN, T., AND INKPEN, K. M. Gathering evidence: Use of visual security cues in web browsers. In *Proceedings of Graphics Interface 2005* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005), GI '05, Canadian Human-Computer Communications Society, pp. 137–144.

[131] WILSON, K. Bug 523652 - IPS action items re IPS SERVIDORES root certificate, Nov. 2009. https://bugzilla.mozilla.org/show_bug.cgi?id=523652.

[132] YILEK, S., RESCORLA, E., SHACHAM, H., ENRIGHT, B., AND SAVAGE, S. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In *2009 ACM SIGCOMM Internet Measurement Conference*, pp. 15–27.

[133] ZHANG, L., ESTRIN, D., BURKE, J., JACOBSON, V., THORNTON, J. D., SMETTERS, D. K., ZHANG, B., TSUDIK, G., MASSEY, D., PAPADOPOULOS, C., ET AL. Named data networking (NDN) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* (2010).

[134] ZUSMAN, M. Breaking the Security Myths of Extended Validation SSL Certificates. In *Black Hat USA 2009* (2009). http://www.blackhat.com/presentations/bh-usa-09/ZUSMAN/BHUSA09-Zusman-AttackExtSSL-SLIDES.pdf.

[135] ZUSMAN, M. Criminal charges are not pursued: Hacking PKI, Aug. 2009. DefCon 17.