Fast Internet-Wide Scanning: A New Security Perspective

by

Zakir Durumeric

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
2017

Doctoral Committee:

Professor J. Alex Halderman, Chair
Associate Professor Michael Bailey, University of Illinois Urbana-Champaign
Research Professor Peter Honeyman
Professor Vern Paxson, University of California, Berkeley
Assistant Professor Florian Schaub

Zakir Durumeric

zakir@umich.edu

ORCID iD: 0000-0002-9647-4192

ACKNOWLEDGMENTS

I want to thank my advisor J. Alex Halderman, as well as Michael Bailey and Vern Paxson, for their support and guidance over the past six years. I have learned an immense amount both professionally and personally from each of you, and I would not be where I am today without you. I also thank my two other committee members—Peter Honeyman and Florian Schaub—as well as Prabal Dutta and Brian Noble for their time and advice.

I would not have been able to complete this dissertation without the help of family and friends. I want to thank my parents Oguz Durumeric and Robin Paetzold, and my brother Aleksander Durumeric for their unwavering support. I am further grateful for the friends who have kept me sane: Brad Campbell, Noah Klugman, Kevin Lu, Corey Melnick, Pat Pannuto, Bethany Patten, Lane Powell, and Audrey Shelton.

I thank my labmates and others in the department for their camaraderie and discussion: David Adrian, Jethro Beekman, Matthew Bernhard, Denis Bueno, Mike Chow, Meghan Clark, Rob Cohn, Jakub Czyz, David Devecsery, Chris Dzombak, Denis Foo Kune, Branden Ghena, Grant Ho, Will Huang, James Kasten, Deepak Kumar, Kyle Lady, Frank Li, Zane Ma, Bill Marczak, Allison McDonald, Ariana Mirian, Paul Pearce, Drew Springall, Benjamin VanderSloot, Eric Wustrow, and Jing Zhang. I am particularly grateful to David Adrian, who has helped immensely on a number of projects over the past three years.

Last, I want to thank the exceptional staff at the University of Michigan and Merit Network for their continuing help. This dissertation would not have been possible without Joe Adams, Kyle Banas, Sol Bermann, Jack Bernard, Chris Brenner, Kevin Cheek, Kelly Cormier, Laura Fink, Paul Howell, Michalis Kallitsis, Dan Maletta, Charlie Mattison, Steve Reger, Will Rhee, Don Winsor, and many others from ITS, CAEN, and DCO.

The work in this dissertation was supported in part by the National Science Foundation, by the Google Anti-abuse Team, and by a Google Ph.D. Fellowship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Techniques like passive observation and random sampling let researchers understand many aspects of Internet day-to-day operation, yet these methodologies often focus on popular services or a small demographic of users, rather than providing a comprehensive view of the devices and services that constitute the Internet. As the diversity of devices and the role they play in critical infrastructure increases, so does understanding the dynamics of and securing these hosts. This dissertation shows how fast Internet-wide scanning provides a near-global perspective of edge hosts that enables researchers to uncover security weaknesses that only emerge at scale.

First, I show that it is possible to efficiently scan the IPv4 address space. ZMap: a network scanner specifically architected for large-scale research studies can survey the entire IPv4 address space from a single machine in under an hour at 97% of the theoretical maximum speed of gigabit Ethernet with an estimated 98% coverage of publicly available hosts. Building on ZMap, I introduce Censys, a public service that maintains up-to-date and legacy snapshots of the hosts and services running across the public IPv4 address space. Censys enables researchers to efficiently ask a range of security questions.

Next, I present four case studies that highlight how Internet-wide scanning can identify new classes of weaknesses that only emerge at scale, uncover unexpected attacks, shed light on previously opaque distributed systems on the Internet, and understand the impact of consequential vulnerabilities. Finally, I explore how increased contention over IPv4 addresses introduces new challenges for performing large-scale empirical studies. I conclude with suggested directions that the research community needs to consider to retain the degree of visibility that Internet-wide scanning currently provides.

Thesis Statement: Fast Internet-wide scanning enables researchers to identify new classes of weaknesses that only emerge at scale, uncover unexpected attacks, shed light on previously opaque distributed systems on the Internet, and understand the impact of consequential vulnerabilities.

CHAPTER 1

Introduction

The Internet has fundamentally transformed communication and the spread of ideas. It is the cornerstone of modern commerce, and critical physical infrastructure is becoming increasingly Internet-controlled. Despite our increasing reliance on the Internet, understanding the dynamics of and securing the devices that constitute it has remained an elusive goal.

One of the reasons that this has remained a challenge is the number of confounding factors that affect device vulnerability. Operators make unexpected errors when configuring networks and servers [97, 177]; miscommunication occurs between scientists, standards bodies, and implementers [23, 232]; users misunderstand warnings and make counterintuitive decisions [28, 108]; manufacturers optimize for cost and performance over security [34, 133, 180, 231]; and even the best secured networks can be taken offline based on the arbitrary decisions of attackers [34, 146]. This complex interplay has rendered deductive reasoning alone about the security of the Internet largely impractical.

Instead, researchers turn to empirical analysis to uncover how devices interact at scale, identify factors that lead to compromise, and understand user, manufacturer, and attacker behavior. Empirical analysis is complicated by the distributed nature of the Internet. By design, there is no central management or coordination of the Internet, let alone visibility into the configuration and vulnerability of individual organizations and devices. Instead, the research community has introduced a variety of measurement techniques that shed light on different aspects of Internet operation. These include both active and passive methodologies. For example, researchers often passively observe network traffic at regional service providers to understand client configuration, analyze probes received in large swaths of unused IP space to understand bot infections, and collect routing data to understand Internet topology.

To track the configuration, patching, and vulnerability of edge hosts, researchers commonly use active scanning—probing large random subsets of the public IP address space to gather presence and configuration data. The research community has used active scanning to understand IPv4 depletion [130], monitor security mitigations [84, 208], and shed light

on previously opaque distributed ecosystems [102]. Unfortunately, while it is possible to collect meaningful data through scanning, probing the entire public address space with existing tools has been prohibitively difficult and slow prior to 2013. Only a handful of full scans were successfully completed since 1982 [84, 102, 130, 163, 208]. Instead, researchers commonly scanned a subset of popular services or random samples of the Internet.

While these restricted approaches allow researchers to understand subsets of the Internet, they do not provide a comprehensive view of the plethora of devices and services that now constitute the Internet. Beyond understanding the dynamics of individual devices, recent research has shown that certain security properties can be easily observed only using a near-global perspective [133]. In this dissertation, I show that it is possible to quickly scan the full public IPv4 address space from a single computer, greatly reducing the barrier to entry for researchers to comprehensively understand the edge hosts on the Internet. I then show how fast Internet-wide scanning enables researchers to identify weaknesses that only emerge at scale, uncover unexpected attacks, shed light on previously opaque distributed systems, and understand the impact of consequential vulnerabilities.

1.1 ZMap: A Fast Internet Scanner

I first introduce ZMap: an open-source network scanner designed to perform large-scale research studies. ZMap is capable of surveying the entire public IPv4 address space in under an hour from a single unmodified desktop machine. Compared to Nmap—an excellent general-purpose network scanner and the industry standard—ZMap achieves much higher performance and coverage. It completes scans more than 1300 times faster than Nmap's most aggressive settings and with 15% higher coverage of listening hosts.

ZMap is not the first system that has been used for performing full scans of the Internet. Researchers have previously used both standard tools like Nmap [102, 133] and custom-designed research scanners [84, 130]. However, these systems have required massive parallelization [133], complex OS modification [163], or months to complete [84, 102, 130]. By increasing scan efficiency and lowering deployment complexity, ZMap dramatically reduces the barrier of entry for performing Internet-wide scans. ZMap's performance further reduces the inaccuracies that arise from DHCP churn and enables researchers to understand the dynamics of quickly changing ecosystems.

ZMap's performance gains are due to several architectural choices:

Stateless and Asynchronous Packet Transmission While previous scanners maintained per-connection state in order to track which hosts have been scanned and to handle timeouts and retransmissions, ZMap forgoes all per-connection state. Instead, ZMap selects addresses according to a random permutation generated by a cyclic multiplicative group. To validate

the responses it receives, ZMap overloads unused values in each probe packet with host-specific cryptographic secrets, similar to SYN cookies [48].

Avoiding Path Congestion Unlike previous scanners, which adapted their transmission rate to avoid saturating source or destination networks, we assume that the source network is well provisioned and ensure that target hosts are randomly ordered and widely dispersed in order to prevent overwhelming destination networks. We send probes as quickly as the scanner's NIC can support, bypassing the operating system's TCP/IP stack, and generating Ethernet frames directly and efficiently.

These design choices enable ZMap to send packets at 97% of the theoretical speed of Gigabit Ethernet. We experimentally analyze ZMap's performance and accuracy. We show that there is no statistical correlation between scan rate and the hosts that ZMap detects when scanning the public IP space at full speed, and that ZMap achieves an estimated 98% coverage of responsive IPv4 hosts. We explore the security implications of high-speed Internet-scale network surveys, both offensive and defensive and discussed best practices for good Internet citizenship when performing Internet-wide surveys.

1.2 Censys: Facilitating Easy Access to Scan Data

While ZMap drastically reduces the time required to conduct large-scale transport-layer scans, collecting meaningful application-layer data through Internet-wide scanning is a specialized and labor-intensive process. Answering simple questions can take weeks of implementing and debugging an application-layer scanner on top of negotiating with institutional legal and networking teams. For these reasons, Internet-wide scanning has remained the province of a small number of research groups—severely limiting its applications. To make Internet-wide scanning available to a wider audience of researchers and security practitioners, I present Censys: a public service that maintains up-to-date and legacy snapshots of the hosts and services running on the public IPv4 address space.

To approximate a real-time "bird's eye view" of the Internet, Censys continually scans the public address space across a range of important ports and protocols. It validates this data and performs application-layer handshakes using a pluggable scanner framework, which dissects handshakes to produce structured data about each host and protocol. The resulting data is post-processed with an extensible annotation framework that enables researchers to programmatically define additional attributes that identify device models and tag security-relevant properties.

Censys exposes data to researchers through a public search interface, SQL engine, and downloadable datasets. The search interface supports full-text searches and querying the structured fields and tags produced during scanning and post processing. Censys returns

results in sub-second time and users can interactively explore the hosts, websites, certificates, and networks that match their query, as well as generate statistical reports suitable for direct use in research.

By moving ZMap scanning to the cloud, Censys further reduces the effort needed to test simple security hypotheses and answer common questions about the composition of the Internet. The continuous data collection provided by Censys additionally facilitates tracking the Internet's evolution over time. This enables researchers to focus on more important questions rather than on the mechanics of answering them, as well as to more efficiently iterate over follow-up questions.

1.3 Applications of High-Speed Scanning

In the second half of my dissertation, I show how the perspective provided by Internet-wide scanning enables researchers to identify new classes of security weaknesses, gain visibility into previously opaque distributed systems, uncover unexpected attacks, and track the impact of vulnerabilities by examining four case studies based on Internet-wide scanning.

1.3.1 Detecting Widespread Weak Keys in Network Devices

In my first case study, I use Internet-wide scanning to show that there exists a class of security phenomena that can be easily observed only from a near global perspective.

Randomness is essential for modern cryptography, and security protections can fail catastrophically when used with a malfunctioning random number generator (RNG). We conduct the first Internet-wide survey to detect cryptographic keys compromised by weak RNGs. We find that compromised keys are surprisingly widespread: 0.75% of TLS certificates share keys due to insufficient entropy, and another 1.70% are generated by the same faulty implementations and thus susceptible to compromise. More alarmingly, we are able to compute the RSA private keys for 0.5% of TLS hosts because their public keys shared non-trivial common factors and the DSA private keys for 1.0% of SSH hosts due to insufficient signature randomness.

We investigate the vulnerable hosts and find that the vast majority are headless or embedded devices. Every device we examine relies on `/dev/urandom` to generate cryptographic keys. By analyzing the RNG's behavior in a laboratory environment, we uncover a boot-time entropy hole that causes urandom to produce deterministic output on headless and embedded devices. We identify vulnerable devices from 55 manufacturers, including some of the largest names in the technology industry. To address the vulnerability, we worked with the responsible parties to correct the flaws, and patch the Linux kernel.

These vulnerabilities are rooted in low-level code in the Linux kernel, and many of

the collisions we found are too rare to ever have been observed in a local environment or through random sampling. In some cases, only two hosts shared a key or factor. However, with the near-global view of the universe of keys provided by Internet-wide scanning, these vulnerabilities quickly became apparent.

1.3.2 Analyzing the HTTPS Certificate Ecosystem

In the second case study, I show how Internet-wide scanning can shed light on previously opaque distributed systems by analyzing the HTTPS public key infrastructure (PKI).

Nearly all secure web communication takes place over HTTPS, including online banking, email, and ecommerce transactions. HTTPS is based on the TLS encrypted transport protocol and a supporting PKI composed of hundreds of certificate authorities (CAs)—entities that are trusted by users' browsers to vouch for the identity of web servers. We place our full trust in each of these authorities—*every* CA has the ability to sign trusted certificates for any domain. Therefore the entire PKI is only as secure as the weakest CA. Nevertheless, this critical infrastructure is strikingly opaque. There is no published list of signed website certificates, or even of the organizations that have trusted signing ability.

To understand the ecosystem, we performed 110 exhaustive IPv4 scans of HTTPS hosts over a 14 month period. We collect 42 million certificate chains, which we use to identify the organizations that control CA certificates and analyze how site certificates are used in practice. We identify 1,832 signing certificates controlled by 683 organizations. More than 80% of these organizations are not commercial certificate authorities. Rather, these CA certificates belong to religious groups, museums, libraries, and more than 130 corporations and financial institutions. None of these CA certificates are constrained and can sign certificates for any website. We investigate how organizations are acquiring signing certificates and identify two sets of accidentally issued signing certificates.

While there are a large number of CAs, the distribution of certificates among authorities is heavily skewed towards a small handful of companies. Three organizations control 75% of all trusted certificates. Worryingly, the compromise of a single private key would require 26% of HTTPS websites on the Internet to immediately obtain new certificates. We provide an up-to-date analysis of the cryptographic algorithms used to sign leaf certificates, and note that half of trusted site certificates contain an inadequately secure 1024-bit RSA key in their trust chain. Last, we find 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.

While it was widely recognized that there were problems in the PKI and that CAs were abusing their power, Internet-wide scanning allowed us to formally identify these organizations, document worrisome practices, and uncover misissued certificates for one of

the first times. ZMap's performance allowed us to perform regular, bidiurnal scans during our study period. This increased our visibility into smaller CAs that were not present in every scan. We specifically identified 376% more leaf certificates and 24% more CA certificates than a prior study that completed a single scan over the course of three months [102]. Frequent scans also enabled us to to track changes to the ecosystem over the course of the year, and we note significant changes in market share. In the most significant example, StartCom usage grew by more than 30%.

1.3.3 Uncovering Attacks on Email Delivery

In my third case study, I show how Internet-wide scanning can help uncover evidence of new, unforeseen attacks occurring in the wild.

Electronic mail carries some of users' most sensitive communication, including private correspondence and financial details. Users expect that messages are private and unforgeable. However, as originally conceived, SMTP—the protocol responsible for relaying messages between mail servers—does not authenticate senders or encrypt mail in transit. Instead, these features were later introduced as protocol extensions. Their adoption is voluntary and extensions fail open (sending mail unprotected) to maintain backwards compatibility. Under the assumption that man-in-the-middle attackers do not exist between large Internet service providers, operators have not prioritized their deployment and standards bodies have not hardened extensions to protect against active adversaries.

We measure the global adoption of SMTP security extensions and the resulting impact on users from three perspectives: scans of SMTP servers on the IPv4 address space, SMTP connection logs for Gmail, and a snapshot of SMTP server configurations for the Alexa Top Million domains. We find that best practices have yet to reach widespread adoption and that the fail-open nature of the protocols has led to a fractured ecosystem where senders cannot reliably authenticate destination servers and recipients are not configured to support authentication. By analyzing our scan logs with mail servers on the IPv4 address space, we uncover evidence that network attackers in some large ISPs are blocking the encryption of mail by corrupting STARTTLS handshakes. By correlating this scan data with Gmail logs, we find that more than 20% of inbound Gmail messages arrive in cleartext due to network attackers. In the most extreme case, Tunisia, more than 96% of messages sent to Gmail are downgraded to cleartext.

This case study not only demonstrates how Internet-wide scanning can uncover unforeseen attacks, but additionally highlights how the visibility and interactivity that Censys provides can help unearth security problems. We uncovered the attack while using Censys to explore SMTP deployment, and Censys enabled us to understand how and where the

problem was occurring. Without this visibility into the scan data, the attack would have likely gone unnoticed.

1.3.4 Tracking Vulnerability Impact

In my fourth and last study, I show how fast Internet-wide scanning allows researchers to quickly react to new vulnerabilities and to understand patching behavior.

In March 2014, researchers found a catastrophic vulnerability in OpenSSL, the cryptographic library used to secure connections in popular server products like 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.

We perform a comprehensive analysis of the vulnerability's impact, including (1) tracking the vulnerable population, (2) monitoring patching behavior over time, (3) assessing the impact on the HTTPS certificate ecosystem using Internet-wide scanning. We find that the vulnerability was widespread, and estimate that between 24–55% of the one million most popular sites that support HTTPS were initially vulnerable, including 44 of the top 100 sites. 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, patching plateaued after about two weeks, and 3% of HTTPS sites in the top million remained vulnerable nearly two months after disclosure. 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 investigate the attack landscape, finding no evidence of large-scale attacks prior to the public disclosure, but that vulnerability scans began within 22 hours, before most sites had patched.

We would not have been able to track Heartbleed's impact with traditional tools due to the amount of time they required to complete scans. ZMap allowed us to start regular, comprehensive scans within 48 hours of disclosure, in turn shedding light on how organizations react to catastrophic vulnerability.

Together, these four case studies highlight how fast Internet-wide scanning enables researchers to uncover security weaknesses and understand the dynamics of edge hosts that emerge only at scale.

1.3.5 Dissertation Roadmap

This remainder of this dissertation is composed of seven chapters. In Chapter 2, I present ZMap's architecture, experimentally characterize its performance and accuracy, and explore

the security implications of high speed Internet-scale network surveys. Then, I introduce Censys, a public search engine and data processing facility backed by data collected from ongoing Internet-wide scans in Chapter 3. In Chapters 4–7, I present four case studies that highlight how Internet-wide scanning enables researchers to uncover and track a range of security weaknesses. Last, I conclude in Chapter 8, with possible research directions to support future empirical security studies.

CHAPTER 2

Fast Internet-Wide Scanning

2.1 Introduction

Internet-scale network surveys collect data by probing large subsets of the public IP address space. While such scanning behavior is often associated with botnets and worms, it also has proved to be a valuable methodology for security research. Recent studies have demonstrated that Internet-wide scanning can help reveal new kinds of vulnerabilities, monitor deployment of mitigations, and shed light on previously opaque distributed ecosystems [102, 130, 133, 137, 185, 208]. Unfortunately, this methodology has been more accessible to attackers than to legitimate researchers, who cannot employ stolen network access or spread self-replicating code. Comprehensively scanning the public address space with off-the-shelf tools like Nmap [168] requires weeks of time or many machines.

In this chapter, we introduce ZMap, a modular and open-source network scanner specifically designed for performing comprehensive Internet-wide research scans. A single mid-range machine running ZMap is capable of scanning for a given open port across the entire public IPv4 address space in under 45 minutes—over 97% of the theoretical maximum speed of gigabit Ethernet—without requiring specialized hardware [128] or kernel modules [86, 219]. ZMap's modular architecture can support many types of single-packet probes, including TCP SYN scans, ICMP echo request scans, and application-specific UDP scans, and it can interface easily with user-provided code to perform follow-up actions on discovered hosts, such as completing a protocol handshake.

Compared to Nmap—an excellent general-purpose network mapping tool, which was utilized in recent Internet-wide survey research [102, 133]—ZMap achieves much higher performance for Internet-scale scans. Experimentally, we find that ZMap is capable of scanning the IPv4 public address space over 1300 times faster than the most aggressive Nmap default settings, with equivalent accuracy. These performance gains are due to architectural choices that are specifically optimized for this application:

Optimized probing While Nmap adapts its transmission rate to avoid saturating the source or target networks, we assume that the source network is well provisioned (unable to be saturated by the source host), and that the targets are randomly ordered and widely dispersed (so no distant network or path is likely to be saturated by the scan). Consequently, we attempt to send probes as quickly as the source's NIC can support, skipping the TCP/IP stack and generating Ethernet frames directly. We show that ZMap can send probes at gigabit line speed from commodity hardware and entirely in user space.

No per-connection state While Nmap maintains state for each connection to track which hosts have been scanned and to handle timeouts and retransmissions, ZMap forgoes any per-connection state. Since it is intended to target random samples of the address space, ZMap can avoid storing the addresses it has already scanned or needs to scan and instead selects addresses according to a random permutation generated by a cyclic multiplicative group. Rather than tracking connection timeouts, ZMap accepts response packets with the correct state fields for the duration of the scan, allowing it to extract as much data as possible from the responses it receives. To distinguish valid probe responses from background traffic, ZMap overloads unused values in each sent packet, in a manner similar to SYN cookies [48].

No retransmission While Nmap detects connection timeouts and adaptively retransmits probes that are lost due to packet loss, ZMap (to avoid keeping state) always sends a fixed number of probes per target and defaults to sending only one. In our experimental setup, we estimate that ZMap achieves 98% network coverage using only a single probe per host, even at its maximum scanning speed.

We further describe ZMap's architecture and implementation in Section 2.2, and we experimentally characterize its performance in Section 2.3. In Section 2.4, we investigate the implications of the widespread availability of fast, low-cost Internet-wide scanning for both defenders and attackers, and we demonstrate ZMap's performance and flexibility in a variety of security settings, including:

1. *Measuring protocol adoption*, such as the transition from HTTP to HTTPS. We explore HTTPS adoption based on frequent Internet-wide scans over a year.
2. *Visibility into distributed systems*, such as the certificate authority (CA) ecosystem. We collect and analyze TLS certificates and identify misissued CA certs.
3. *High-speed vulnerability scanning*, which could allow attackers to widely exploit vulnerabilities within hours of their discovery. We build a UPnP scanner using ZMap through which we find 3.4 million UPnP devices with known vulnerabilities [185].
4. *Uncovering unadvertised services*, such as hidden Tor bridges. We show that ZMap can locate 86% of hidden Tor bridges via comprehensive enumeration.

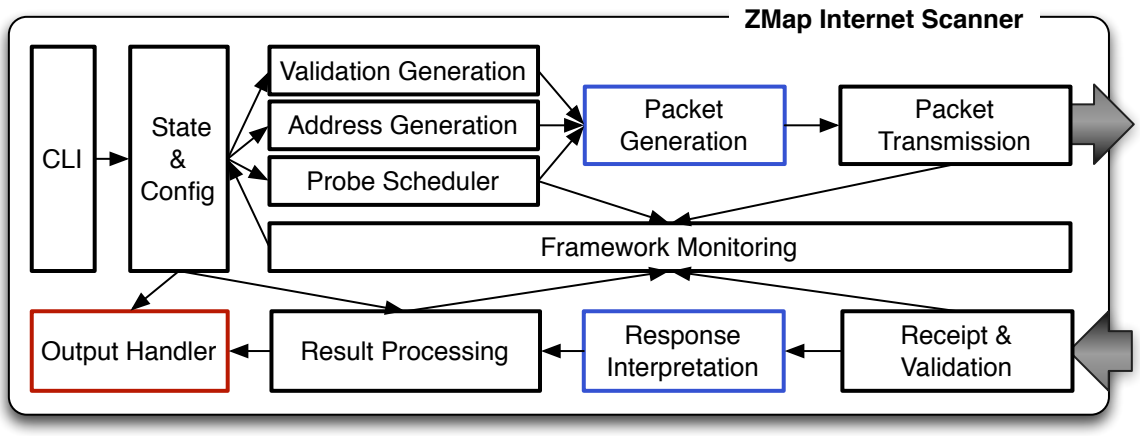


Figure 2.1: **ZMap Architecture** — ZMap is an open-source network scanner optimized for efficiently performing Internet-scale network surveys. Modular packet generation and response interpretation components (*blue*) support multiple kinds of probes, including TCP SYN scans and ICMP echo scans. Modular output handlers (*red*) allow users to output or act on scan results in application-specific ways. The architecture allows sending and receiving components to run asynchronously and enables a single source machine to comprehensively scan every host in the public IPv4 address space for a particular open TCP port in under 45 mins using a 1 Gbps Ethernet link.

High-speed scanning can be a powerful tool in the hands of security researchers, but users must be careful not to cause harm by inadvertently overloading networks or causing unnecessary work for network administrators. In Section 7.7, we discuss our experiences performing numerous large-scale scans over the past year, we report on the complaints and other reactions we have received, and we suggest several guidelines and best practices for good Internet citizenship while scanning.

Internet-wide scanning has already shown great potential as a research methodology [102, 130, 133, 185], and we hope ZMap will facilitate a variety of new applications by drastically reducing the costs of comprehensive network surveys and allowing scans to be performed with very fine time granularity. To facilitate this, we are releasing ZMap as an open source project that is documented and packaged for real world use. It is available at https://zmap.io/.

2.2 ZMap: The Scanner

ZMap uses a modular design to support many types of probes and integration with a variety of research applications, as illustrated in Figure 2.1. The *scanner core* handles command line and configuration file parsing, address generation and exclusion, progress and performance monitoring, and reading and writing network packets. Extensible *probe modules* can be customized for different kinds of probes, and are responsible for generating probe packets

and interpreting whether incoming packets are valid responses. Modular *output handlers* allow scan results to be piped to another process, added directly to a database, or passed on to user code for further action, such as completing a protocol handshake.

We introduced the philosophy behind ZMap's design in Section 2.1. At a high level, one of ZMap's most important architectural features is that sending and receiving packets take place in separate threads that act independently and continuously throughout the scan. A number of design choices were made to ensure that processes share as little state as possible. We implemented ZMap in approximately 8,900 SLOC of C.

2.2.1 Addressing Probes

If ZMap simply probed every IPv4 address in numerical order, it would risk overloading destination networks with scan traffic and produce inconsistent results in the case of a distant transient network failure. To avoid this, ZMap scans addresses according to a random permutation of the address space. To select smaller random samples of the address space, we simply scan a subset of the full permutation.

ZMap uses a simple and inexpensive method to traverse the address space, which lets it scan in a random permutation while maintaining only negligible state. We iterate over a multiplicative group of integers modulo p, choosing p to be a prime slightly larger than 2^{32}. By choosing p to be a prime, we guarantee that the group is cyclic and will reach all addresses in the IPv4 address space except 0.0.0.0 (conveniently an IANA reserved address) once per cycle. We choose to iterate over $(\mathbb{Z}/4,294,967,311\mathbb{Z})^{\times}$, the multiplicative group modulo p for the smallest prime larger than 2^{32}: $2^{32}+15$.

To select a fresh random permutation for each scan, we generate a new primitive root of the multiplicative group and choose a random starting address. Because the order of elements in a group is preserved by an isomorphism, we efficiently find random primitive roots of the multiplicative group by utilizing the isomorphism $(\mathbb{Z}_{p-1},+) \cong (\mathbb{Z}_p^*,\times)$ and mapping roots of $(\mathbb{Z}_{p-1},+)$ into the multiplicative group via the function $f(x) = n^x$ where n is a known primitive root of $(\mathbb{Z}/p\mathbb{Z})^{\times}$. In our specific case, we know that 3 is a primitive root of $(\mathbb{Z}/4,294,967,311\mathbb{Z})^{\times}$.

Because we know that the generators of $(\mathbb{Z}_{p-1},+)$ are $\{s|(s,p-1)=1\}$, we can efficiently find the generators of the additive group by precalculating and storing the factorization of $p-1$ and checking addresses against the factorization at random until we find one that is coprime with $p-1$ and then map it into $(\mathbb{Z}_p^*,\times)$. Given that there exist approximately 10^9 generators, we expect to make four tries before finding a primitive root. While this process introduces complexity at the beginning of a scan, it adds only a small amount of one-time overhead.

Once a primitive root has been generated, we can easily iterate through the address space by applying the group operation to the current address (in other words, multiplying the current address by the primitive root modulo $2^{32}+15$). We detect that a scan has completed when we reach the initially scanned IP address. This technique allows the sending thread to store the selected permutation and progress through it with only three integers: the primitive root used to generate the multiplicative group, the first scanned address, and the current address.

Excluding Addresses Since ZMap is optimized for Internet-wide scans, we represent the set of targets as the full IPv4 address space minus a set of smaller excluded address ranges. Certain address ranges need to be excluded for performance reasons (e.g., skipping IANA reserved allocations) and others to honor requests from their owners to discontinue scanning. We efficiently support address exclusion through the use of radix trees, a trie specifically designed to handle ranges and frequently used by routing tables [227, 233]. Excluded ranges can be specified through a configuration file.

2.2.2 Packet Transmission and Receipt

ZMap is optimized to send probes as quickly as the source's CPU and NIC can support. The packet generation component operates asynchronously across multiple threads, each of which maintains a tight loop that sends Ethernet-layer packets via a raw socket.

We send packets at the Ethernet layer in order to cache packet values and reduce unnecessary kernel overhead. For example, the Ethernet header, minus the packet checksum, will never change during a scan. By generating and caching the Ethernet layer packet, we prevent the Linux kernel from performing a routing lookup, an arpcache lookup, and netfilter checks for every packet. An additional benefit of utilizing a raw socket for TCP SYN scans is that, because no TCP session is established in the kernel, upon receipt of a TCP SYN-ACK packet, the kernel will automatically respond with a TCP RST packet, closing the connection. ZMap can optionally use multiple source addresses and distribute outgoing probes among them in a round-robin fashion.

We implement the receiving component of ZMap using libpcap [144], a library for capturing network traffic and filtering the received packets. Although libpcap is a potential performance bottleneck, incoming response traffic is a small fraction of outgoing probe traffic, since the overwhelming majority of hosts are unresponsive to typical probes, and we find that libpcap is easily capable of handling response traffic in our tests (see Section 2.3). Upon receipt of a packet, we check the source and destination port, discard packets clearly not initiated by the scan, and pass the remaining packets to the active probe module for interpretation.

While the sending and receiving components of ZMap operate independently, we ensure that the receiver is initialized prior to sending probes and that the receiver continues to run for a period of time (by default, 8 seconds) after the sender has completed in order to process any delayed responses.

2.2.3 Probe Modules

ZMap probe modules are responsible for filling in the body of probe packets and for validating whether incoming packets are responsive to the probes. Making these tasks modular allows ZMap to support a variety of probing methods and protocols and simplifies extensibility. Out of the box, ZMap provides probe modules to support TCP port scanning and ICMP echo scanning.

At initialization, the scanner core provides an empty buffer for the packet and the probe module fills in any static content that will be the same for all targets. Then, for each host to be scanned, the probe module updates this buffer with host-specific values. The probe module also receives incoming packets, after high-level validation by the scanner core, and determines whether they are positive or negative responses to scan probes. Users can add new scan types by implementing a small number of callback functions within the probe module framework.

For example, to facilitate TCP port scanning, ZMap implements a probing technique known as *SYN scanning* or *half-open scanning*. We chose to implement this specific technique instead of performing a full TCP handshake based on the reduced number of exchanged packets. In the dominant case where a host is unreachable or does not respond, only a single packet is used (a SYN from the scanner); in the case of a closed port, two packets are exchanged (a SYN answered with a RST); and in the uncommon case where the port is open, three packets are exchanged (a SYN, a SYN-ACK reply, and a RST from the scanner).

Checking Response Integrity ZMap's receiving components need to determine whether received packets are valid responses to probes originating from the scanner or are part of other background traffic. Probe modules perform this validation by encoding host- and scan-invocation–specific data into mutable fields of each probe packet, utilizing fields that will have recognizable effects on fields of the corresponding response packets in a manner similar to SYN cookies [48].

For each scanned host, ZMap computes a MAC (message authentication code) of the destination address keyed by a scan-specific secret. This MAC value is then spread across any available fields by the active probe module. We chose to use the UMAC function for these operations, based on its performance guarantees [54]. In our TCP port scan module,

we utilize the source port and initial sequence number; for ICMP, we use the ICMP identifier and sequence number. These fields are checked on packet receipt by the probe module, and ZMap discards any packets for which validation fails.

These inexpensive checks prevents the incorrect reporting of spurious response packets due to background traffic as well as responses triggered by previous scans. This design ultimately allows the receiver to validate responses while sharing only the scan secret and the initial configuration with the sending components.

2.2.4 Output Modules

ZMap provides a modular output interface that allows users to output scan results or act on them in application-specific ways. Output module callbacks are triggered by specific events: scan initialization, probe packet sent, response received, regular progress updates, and scan termination. ZMap's built-in output modules cover basic use, including simple text output (a file stream containing a list of unique IP addresses that have the specified port open), extended text output (a file stream containing a list of all packet responses and timing data), and an interface for queuing scan results in a Redis in-memory database [220].

Output modules can also be implemented to trigger network events in response to positive scan results, such as completing an application-level handshake. For TCP SYN scans, the simplest way to accomplish this is to create a fresh TCP connection with the responding address; this can be performed asynchronously with the scan and requires no special kernel support.

2.3 Validation and Measurement

We performed a series of experiments to characterize the performance of ZMap. Under our test setup, we find that a complete scan of the public IPv4 address space takes approximately 44 minutes on an entry-level server with a gigabit Ethernet connection. We estimate that a single-packet scan can detect approximately 98% of instantaneously listening hosts, and we measure a 1300 x performance improvement over Nmap for Internet-wide scanning, with equivalent coverage.

We performed the following measurements on an HP ProLiant DL120 G7 with a Xeon E3-1230 3.2 GHz processor and 4 GB of memory running a stock install of Ubuntu 12.04.1 LTS and the 3.2.0-32-generic Linux kernel. Experiments were conducted using the onboard NIC, which is based on the Intel 82574L chipset and uses the stock e1000e network driver, or a quad-port Intel Ethernet adapter based on the newer Intel 82580 chipset and using the stock igb network driver. For experiments involving complete TCP handshakes, we disabled kernel modules used by iptables and conntrack. Experiments comparing ZMap

with Nmap were conducted with Nmap 5.21.

These measurements were conducted using the normal building network at the University of Michigan Computer Science & Engineering division. We used a gigabit Ethernet uplink (a standard office network connection in our facility); we did not arrange for any special network configuration beyond static IP addresses. 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. We note that ZMap's performance on other source networks may be worse than reported here due to local congestion.

2.3.1 Scan Rate: How Fast is Too Fast?

In order to determine whether our scanner and our upstream network can handle scanning at gigabit line speed, we examine whether the *scan rate*, the rate at which ZMap sends probe packets, has any effect on the *hit rate*, the fraction of probed hosts that respond positively (in this case, with a SYN-ACK). If libpcap, the Linux kernel, our institutional network, or our upstream provider are unable to adequately handle the traffic generated by the scanner at full speed, we would expect packets to be dropped and the hit rate to be lower than at slower scan rates.

We experimented by sending TCP SYN packets to random 1% samples of the IPv4 address space on port 443 at varying scan rates. We conducted 10 trials at each of 16 scan rates ranging from 1,000 to 1.4 M packets per second. The results are shown in Figure 2.2.

We find no statistically significant correlation between scan rate and hit rate. This shows that our ZMap setup is capable of handling scanning at 1.4 M packets per second and that scanning at lower rates provides no benefit in terms of identifying additional hosts. From an architectural perspective, this validates that our receiving infrastructure based on libpcap is capable of processing responses generated by the scanner at full speed and that kernel modules such as PF_RING [86] are not necessary for gigabit-speed network scanning.

2.3.2 Coverage: Is One SYN Enough?

While scanning at higher rates does not appear to result in a lower hit rate, this does not tell us what *coverage* we achieve with a single scan—what fraction of target hosts does ZMap actually find using its default single-packet probing strategy?

Given the absence of ground truth for the number of hosts on the Internet with a specific port open, we cannot measure coverage directly. This is further complicated by the ever changing state of the Internet; it is inherently difficult to detect whether a host was not included in a scan because it was not available at the time or because packets were dropped between it and the scanner. Yet, this question is essential to understanding whether

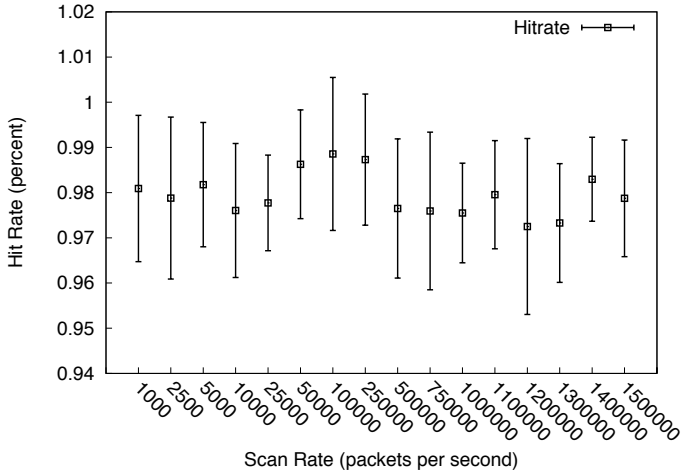


Figure 2.2: **Hit rate vs. Scan rate** — We find no correlation between hit rate (positive responses/hosts probed) and scan rate (probes sent/second). Shown are means and standard deviations over ten trials. This indicates that slower scanning does not reveal additional hosts.

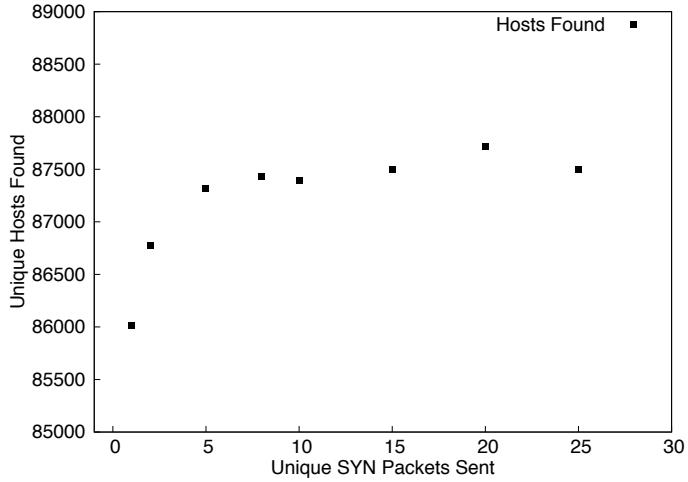


Figure 2.3: **Coverage for Multiple Probes** — Discovered hosts plateau after ZMap sends about 8 SYNs to each. If this plateau represents the true number of listening hosts, sending just 1 SYN will achieve about 98% coverage.

performing fast, single-packet scans is an accurate methodology for Internet-wide surveys.

To characterize ZMap's coverage, we estimate the number of hosts that are actually listening by sending multiple, distinct SYN packets to a large address sample and analyzing the distribution of the number of positive responses received compared to the number of SYNs we send. We expect to eventually see a plateau in the number of hosts that respond regardless of the number of additional SYNs we send. If this plateau exists, we can treat it as an estimate of the real number of listening hosts, and we can use it as a baseline against which to compare scans with fewer SYN packets.

We performed this experiment by sending 1, 2, 5, 8, 10, 15, 20, and 25 SYN packets to random 1% samples of the IPv4 address space on port 443 and recording the number of distinct hosts that sent SYN-ACK responses in each scan. The results indicate a clear plateau in the number of responsive hosts after sending 8 SYN packets, as shown in Figure 2.3.

Based on the level of this plateau, we estimate that our setup reaches approximately 97.9% of live hosts using a single packet, 98.8% of hosts using two packets, and 99.4% of hosts using three packets. The single packet round-trip loss rate of about 2% is in agreement with previous studies on random packet drop on the Internet [130].

These results suggest that single-probe scans are sufficiently comprehensive for typical research applications. Investigators who require higher coverage can configure ZMap to send multiple probes per host, at the cost of proportionally longer running scans.

2.3.3 Variation by Time of Day

In previous work, Internet-wide scans took days to months to execute, so there was little concern over finding the optimal time of day to perform a scan. However, since ZMap scans can take less than an hour to complete, the question as to the "right time" to perform a scan arises. Are there certain hours of the day or days of the week that are more effective for scanning than others?

In order to measure any diurnal effects on scanning, we performed continuous scans of TCP port 443 targeting a random 1% sample of the Internet over a 24-hour period. Figure 2.4 shows the number of hosts found in each scan.

We observed a $\pm 3.1\%$ variation in hit rate dependent on the time of day scans took place. The highest response rates were at approximately 7:00 AM EST and the lowest response rates were at around 7:45 PM EST.

These effects may be due to variation in overall network congestion and packet drop rates or due to a diurnal pattern in the aggregate availability of end hosts that are only intermittently connected to the network. In less formal testing, we did not notice any obvious variation by day of the week or day of the month.

Scan Type	Coverage (normalized)	Duration (mm:ss)	Estimated Time for Internet-wide Scan
Nmap, max 2 probes (default)	0.978	45:03	116.3 days
Nmap, 1 probe	0.814	24:12	62.5 days
ZMap, 2 probes	1.000	00:11	2:12:35
ZMap, 1 probe (default)	0.987	00:10	1:09:45

Table 2.1: **ZMap vs. Nmap Comparison** — We scanned 1 million hosts on TCP port 443 using ZMap and Nmap and averaged over 10 trials. Despite running hundreds of times faster, ZMap finds more listening hosts than Nmap, due to Nmap's low host timeout. Times for ZMap include a fixed 8 second delay to wait for responses after the final probe.

2.3.4 Comparison with Nmap

We performed several experiments to compare ZMap to Nmap in Internet-wide scanning applications, focusing on coverage and elapsed time to complete a scan. Nmap and ZMap are optimized for very different purposes. Nmap is a highly flexible, multipurpose tool that is frequently used for probing a large number of open ports on a smaller number of hosts, whereas ZMap is optimized to probe a single port across very large numbers of targets. We chose to compare the two because recent security studies used Nmap for Internet-wide surveys [102, 133], and because, like ZMap, Nmap operates from within user space on Linux [168].

We tested a variety of Nmap settings to find reasonable configurations to compare. All performed a TCP SYN scan on port 443 (-Ss -p 443). Nmap provides several defaults known as *timing templates*, but even with the most aggressive of these (labeled "insane"), an Internet-wide scan would take over a year to complete. To make Nmap scan faster in our test configurations, we started with the "insane" template (-T5), disabled host discovery and DNS resolutions (-Pn -n), and set a high minimum packet rate (--min-rate 10000). The "insane" template retries each probe once after a timeout; we additionally tested a second Nmap configuration with retries disabled (--max-retries 0).

We used ZMap to select a random sample of 1 million IP addresses and scanned them for hosts listening on port 443 with Nmap in the two configurations described above and with ZMap in its default configuration and in a second configuration that sends two SYN probes to each host (-P 2). We repeated this process for 10 trials over a 12 hour period and report the averages in Table 2.1.

The results show that ZMap scanned much faster than Nmap and found more listening hosts than either Nmap configuration. The reported durations for ZMap include time sent sending probes as well as a fixed 8-second delay after the sending process completes, during

which ZMap waits for late responses. Extrapolating to the time required for an Internet-wide scan, the fastest tested ZMap configuration would complete approximately 1300 times faster than the fastest Nmap configuration.[1]

Coverage and Timeouts To investigate why ZMap achieved higher coverage than Nmap, we probed a random sample of 4.3 million addresses on TCP port 80 and measured the latency between sending a SYN and receiving a SYN-ACK from responsive hosts. Figure 2.5 shows the CDF of the results. The maximum round-trip time was 450 seconds, and a small number of hosts took more than 63 seconds to respond, the time it takes for a TCP connection attempt to timeout on Linux. 99% of hosts that responded within 500 seconds did so within about 1 second, and 99.9% responded within 8.16 seconds.

As ZMap's receiving code is stateless with respect to the sending code, a valid SYN-ACK that comes back any time before the scan completes will be recorded as a listening host. To assure a high level of coverage, the default ZMap settings incorporate an empirically derived 8-second delay after the last probe is sent before the receiving process terminates.

In contrast, Nmap maintains timeouts for each probe. In the Nmap "insane" timing template we tested, the timeout is initially 250 ms, by which time fewer than 85% of responsive hosts in our test had responded. Over the course of a scan, Nmap's timeout can increase to 300 ms, by which time 93.2% had responded. Thus, we would expect a single-probe Nmap scan with these timing values to see 85–93% of the hosts that ZMap finds, which is roughly in line with the observed value of 82.5%.

With Nmap's "insane" defaults, it will attempt to send a second probe after a timeout. A response to either the first or second SYN will be considered valid until the second times out, so this effectively raises the overall timeout to 500–600 ms, by which time we received 98.2–98.5% of responses. Additional responses will likely be generated by the second SYN. We observed that the 2-probe Nmap scan found 99.1% of the number of hosts that a 1-probe ZMap scan found.

[1] The extrapolated 1-packet Internet-wide scan time for ZMap is longer than the 44 minutes we report elsewhere for complete scans, because this test used a slower NIC based on the Intel 82574L chipset.

Scan	Date	Port 443 Open	TLS Servers	All Certs	Trusted Certs
EFF SSL Observatory [102]	2010/12	16.2 M	7.7 M	4.0 M	1.46 M
Mining Ps and Qs [133]	2011/10	28.9 M	12.8 M	5.8 M	1.96 M
ZMap + certificate fetcher	2012/06	31.8 M	19.0 M	7.8 M	2.95 M
ZMap + certificate fetcher	2013/05	34.5 M	22.8 M	8.6 M	3.27 M

Table 2.2: **Comparison with Prior Internet-wide HTTPS Surveys** — Due to growth in HTTPS deployment, ZMap finds almost three times as many TLS servers as the SSL Observatory did in late 2010, yet this process takes only 10 hours to complete from a single machine using a ZMap-based workflow, versus three months on three machines.

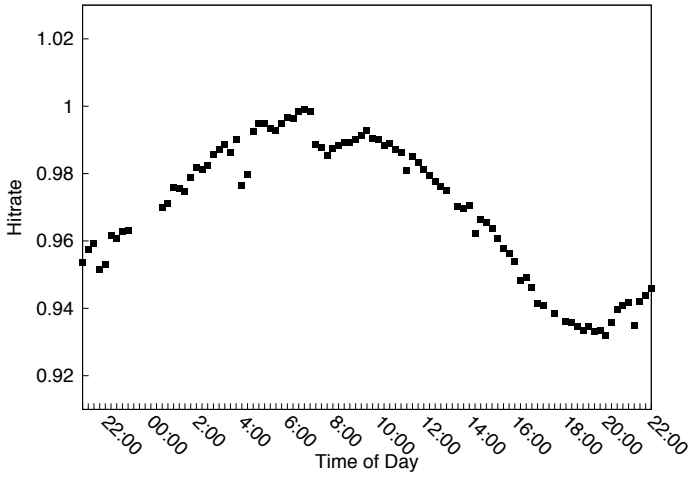


Figure 2.4: **Diurnal Effect on Hosts Found** — We observed a $\pm 3.1\%$ variation in ZMap's hit rate depending on the time of day the scan was performed. (Times EST.)

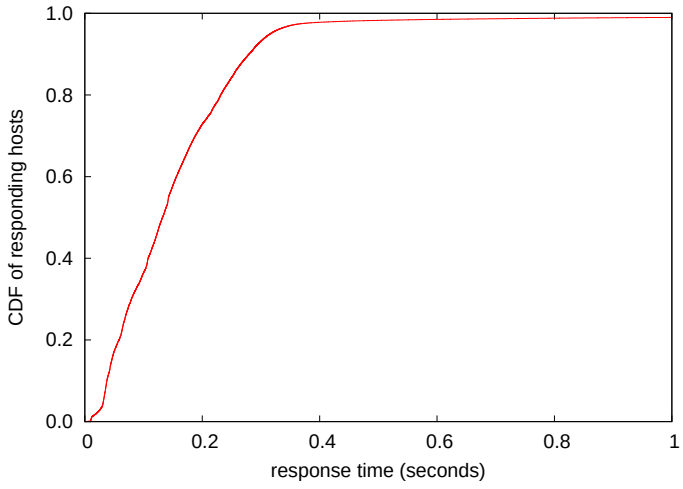


Figure 2.5: **SYN to SYN-ACK time** — In an experiment that probed 4.3 million hosts, 99% of SYN-ACKs arrived within about 1 second and 99.9% within 8.16 seconds.

2.3.5 Comparison with Previous Studies

Several groups have previously performed Internet-wide surveys using various methodologies. Here we compare ZMap to two recent studies that focused on HTTPS certificates. Most recently, Heninger et al. performed a distributed scan of port 443 in 2011 as part of a global analysis on cryptographic key generation [133]. Their scan used Nmap on 25 Amazon EC2 instances and required 25 hours to complete, with a reported average of 40,566 hosts scanned per second. A 2010 scan by the EFF SSL Observatory project used Nmap on 3 hosts and took 3 months to complete [102].

To compare ZMap's performance for this task, we used it to conduct comprehensive scans of port 443 and used a custom certificate fetcher based on libevent [174] and OpenSSL [246] to retrieve TLS certificates from each responsive host. With this methodology, we were able to discover hosts, perform TLS handshakes, and collect and parse the resulting certificates in under 10 hours from a single machine.

As shown in Table 2.2, we find significantly more TLS servers than previous work—78% more than Heninger et al. and 196% more than the SSL Observatory—likely due to increased HTTPS deployment since those studies were conducted. Linear regression shows an average growth in HTTPS deployment of about 540,000 hosts per month over the 29 month period between the SSL Observatory scan and our most recent dataset. Despite this growth, ZMap is able to collect comprehensive TLS certificate data in a fraction of the time and cost needed in earlier work. The SSL Observatory took roughly 650 times as much machine time to acquire the same kind of data, and Heninger et al. took about 65 times as much.

2.4 Applications and Security Implications

The ability to scan the IPv4 address space in under an hour opens an array of new research possibilities, including the ability to gain visibility into previously opaque distributed systems, understand protocol adoption at a new resolution, and uncover security phenomenon only accessible with a global perspective [133]. However, high-speed scanning also has potentially malicious applications, such as finding and attacking vulnerable hosts en masse. Furthermore, many developers have the preconceived notion that the Internet is far too large to be fully enumerated, so the reality of high speed scanning may disrupt existing security models, such as by leading to the discovery of services previously thought to be well hidden. In this section, we use ZMap to explore several of these applications.

2.4.1 Visibility into Distributed Systems

High-speed network scanning provides researchers with the possibility for a new real-time perspective into previously opaque distributed systems on the Internet. For instance, e-

Organization	Certificates	
GoDaddy.com, Inc.	913,416	(31.0%)
GeoTrust Inc.	586,376	(19.9%)
Comodo CA Limited	374,769	(12.7%)
VeriSign, Inc.	317,934	(10.8%)
Thawte, Inc.	228,779	(7.8%)
DigiCert Inc	145,232	(4.9%)
GlobalSign	117,685	(4.0%)
Starfield Technologies	94,794	(3.2%)
StartCom Ltd.	88,729	(3.0%)
Entrust, Inc.	76,929	(2.6%)

Table 2.3: **Top 10 Certificate Authorities** — We used ZMap to perform regular comprehensive scans of HTTPS hosts in order gain visibility into the CA ecosystem. Ten organizations control 86% of browser trusted certificates.

commerce and secure web transactions inherently depend on browser trusted TLS certificates. However, there is currently little oversight over browser trusted certificate authorities (CAs) or issued certificates. Most CAs do not publish lists of the certificates they have signed, and, due to delegation of authority to intermediate CAs, it is unknown what set of entities have the technical ability to sign browser-trusted certificates at any given time.

To explore this potential, we used ZMap and our custom certificate fetcher to conduct regular scans over the past year and perform analysis on new high-profile certificates and CA certificates. Between April 2012 and June 2013, we performed 1.81 billion TLS handshakes, ultimately collecting 33.6 million unique X.509 certificates of which 6.2 million were browser trusted. We found and processed an average of 220,000 new certificates, 15,300 new browser trusted certificates, and 1.2 new CA certificates per scan. In our most recent scan, we identified 1,832 browser trusted signing certificates from 683 organizations and 57 countries. We observed 3,744 distinct browser-trusted signing certificates in total. Table 5.5 shows the most prolific CAs by leaf certificates issued.

Wide-scale visibility into CA behavior can help to identify security problems [102, 147]. We found two cases of misissued CA certificates. In the first case, we found a CA certificate that was accidentally issued to a Turkish transit provider. This certificate, `C=TR, ST=ANKARA, L=ANKARA, O=EGO, OU=EGO BILGI ISLEM, CN=*.EGO.GOV.TR`, was later found by Google after being used to sign a Google wildcard certificate and has since been revoked and blacklisted in common web browsers [156].

In the second case, we found approximately 1,300 CA certificates that were misissued by the Korean Government to government sponsored organizations such as schools and libraries. While these certificates had been issued with rights to sign additional certificates,

a length constraint on the grandparent CA certificate prevented these organizations from signing new certificates. We do not include these Korean certificates in the CA totals above because they are unable to sign valid browser-trusted certificates.

2.4.2 Tracking Protocol Adoption

Researchers have previously attempted to understand the adoption of new protocols, address depletion, common misconfigurations, and vulnerabilities through active scanning [42, 102, 130, 133, 137, 208]. In many of these cases, these analyses have been performed on random samples of the IPv4 address space due to the difficulty of performing comprehensive scans [137, 208]. In cases where full scans were performed, they were completed over an extended period of time or through massive parallelization on cloud providers [102, 133]. ZMap lowers the barriers to entry and allows researchers to perform studies like these in a comprehensive and timely manner, ultimately enabling much higher resolution measurements than previously possible.

To illustrate this application, we tracked the adoption of HTTPS using 158 Internet-wide scans over the past year. Notably, we find a 23% increase in HTTPS use among Alexa Top 1 Million websites and a 10.9% increase in the number of browser-trusted certificates. During this period, the Netcraft Web Survey [188] finds only a 2.2% increase in the number of HTTP sites, but we observe an 8.5% increase in sites using HTTPS. We plot these trends in Figure 2.6.

We can also gain instantaneous visibility into the deployment of multiple protocols by performing many ZMaps scans of different ports. We scanned 0.05% samples of the IPv4 address space on each TCP port below 9175 to determine the percentage of hosts

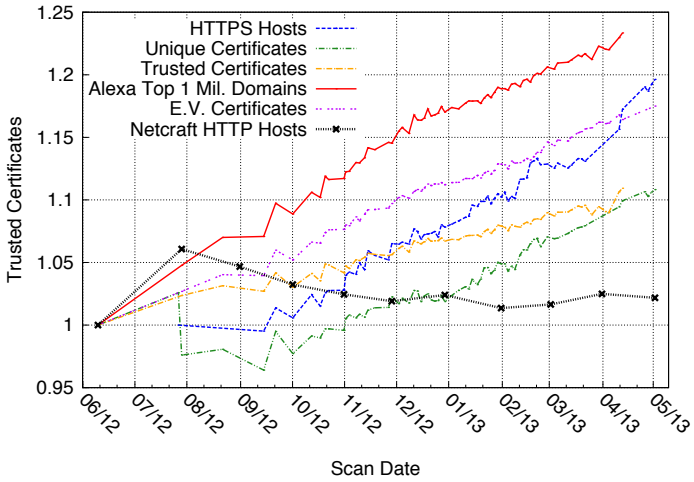


Figure 2.6: **HTTPS Adoption** — Data we collected using ZMap show trends in HTTPS deployment over one year. We observed 19.6% growth in hosts serving HTTPS.

Port	Service	Hit Rate (%)
80	HTTP	1.77
547	CWMP	1.12
443	HTTPS	0.93
21	FTP	0.67
23	Telnet	0.61
22	SSH	0.57
25	SMTP	0.43
3479	2-Wire RPC	0.42
8080	HTTP-alt/proxy	0.38
53	DNS	0.38

Table 2.4: **Top 10 TCP ports** — We scanned 2.15M hosts on TCP ports 0–9175 and observed what fraction were listening on each port. We saw a surprising number of open ports associated with embedded devices, such as TCP/7547 (CWMP).

that were listening on each port. This experiment requires the same number of packets as over 5 Internet-wide scans of a single port, yet we completed it in under a day using ZMap. Table 2.4 shows the top 10 open ports we observed.

2.4.3 Enumerating Vulnerable Hosts

With the ability to perform rapid Internet-wide scans comes the potential to quickly enumerate hosts that suffer from specific vulnerabilities [42]. While this can be a powerful defensive tool for researchers—for instance, to measure the severity of a problem or to track the application of a patch—it also creates the possibility for an attacker with control of only a small number of machines to scan for and infect all public hosts suffering from a new vulnerability within minutes.

UPnP Vulnerabilities To explore these applications, we investigated several recently disclosed vulnerabilities in common UPnP frameworks. On January 29, 2013, HD Moore publicly disclosed several vulnerabilities in common UPnP libraries [185]. These vulnerabilities ultimately impacted 1,500 vendors and 6,900 products, all of which can be exploited to perform arbitrary code execution with a single UDP packet. Moore followed responsible disclosure guidelines and worked with manufacturers to patch vulnerable libraries, and many of the libraries had already been patched at the time of disclosure. Despite these precautions, we found that at least 3.4 million devices were still vulnerable in February 2013.

To measure this, we created a custom ZMap probe module that performs a UPnP discovery handshake. We were able to develop this 150-SLOC module from scratch in approximately four hours and performed a comprehensive scan of the IPv4 address space for publicly available UPnP hosts on February 11, 2013, which completed in under two

hours. This scan found 15.7 million publicly accessible UPnP devices, of which 2.56 million (16.5%) were running vulnerable versions of the Intel SDK for UPnP Devices, and 817,000 (5.2%) used vulnerable versions of MiniUPnPd.[2]

Given that these vulnerable devices can be infected with a single UDP packet [185], we note that these 3.4 million devices could have been infected in approximately the same length of time—much faster than network operators can reasonably respond or for patches to be applied to vulnerable hosts. Leveraging methodology similar to ZMap, it would only have taken a matter of hours from the time of disclosure to infect every publicly available vulnerable host.

Weak Public Keys As part of our regular scans of the HTTPS ecosystem, we tracked the mitigation of the 2008 Debian weak key vulnerability [47] and the weak and shared keys described by Heninger et al. in 2012 [133]. Figure 3.1 shows several trends over the past year.

In our most recent scan, we found that 44,600 unique certificates utilized factorable RSA keys and are served on 51,000 hosts, a 20% decrease from 2011 [133]. Four of these certificates were browser trusted; the last was signed in August 2012. Similarly, we found 2,743 unique certificates that contained Debian weak keys, of which 96 were browser trusted, a 34% decrease from 2011 [133]. The last browser trusted certificate containing a Debian weak key was signed in January 2012. We also observed a 67% decrease in the number of browser-trusted certificates that contained default public keys used for Citrix remote access products [133].

We created an automated process that alerts us to the discovery of new browser-trusted certificates containing factorable RSA keys, Debian weak keys, or default Citrix keys as soon as they are found, so that we can attempt to notify the certificate owners about the vulnerability.

2.4.4 Discovering Unadvertised Services

The ability to perform comprehensive Internet scans implies the potential to uncover unadvertised services that were previously only accessible with explicit knowledge of the host name or address. For example, Tor bridges are intentionally not published in order to prevent ISPs and government censors from blocking connections to the Tor network [238]. Instead, the Tor Project provides users with the IP addresses of a small number of bridges based on their source address. While Tor developers have acknowledged that bridges can in principle be found by Internet-wide scanning [88], the set of active bridges is constantly changing, and

[2]Moore reported many more UPnP hosts [185] but acknowledges that his scans occurred over a 5 month period and did not account for hosts being counted multiple times due to changing IP addresses.

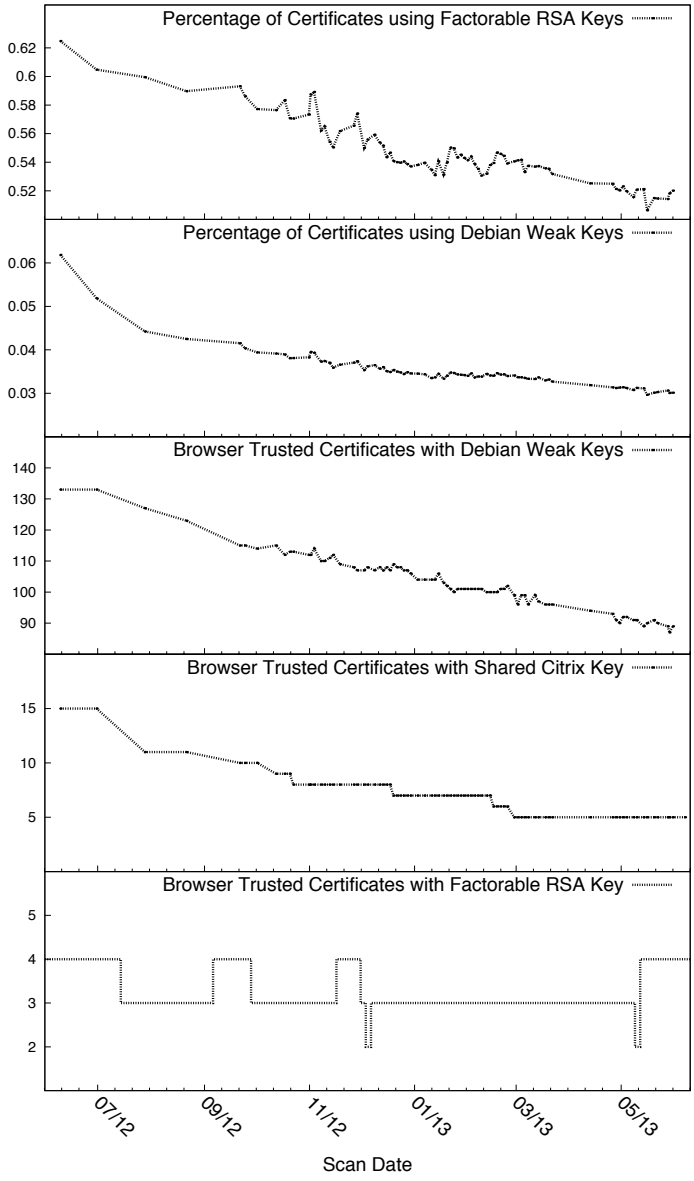


Figure 2.7: **Trends in HTTPS Weak Key Usage** — To explore how ZMap can be used to track the mitigation of known vulnerabilities, we monitored the use of weak HTTPS public keys from May 2012 through June 2013.

the data would be stale by the time a long running scan was complete. However, high-speed scanning might be used to mount an effective attack.

To confirm this, we performed Internet wide-scans on ports 443 and 9001, which are common ports for Tor bridges and relays, and applied a set of heuristics to identify likely Tor nodes. For hosts with one of these ports open, we performed a TLS handshake using a specific set of cipher suites supported by Tor's "v1 handshake." When a Tor relay receives this set of cipher suites, it will respond with a two-certificate chain. The signing ("Certifiate

Authority") certificate is self-signed with the relay's identity public key and uses a subject name of the form "CN=www.*X*.com", where *X* is a randomized alphanumeric string. This pattern matched 67,342 hosts on port 443, and 2,952 hosts on port 9001.

We calculated each host's identity fingerprint and checked whether the SHA1 hash appeared in the public Tor metrics list for bridge pool assignments. Hosts we found matched 1,170 unique bridge fingerprints on port 443 and 419 unique fingerprints on port 9001, with a combined total of 1,534 unique fingerprints (some were found on both ports). From the bridge pool assignment data, we see there have been 1,767–1,936 unique fingerprints allocated at any given time in the recent past, which suggests that we were able to identify 79–86% of allocated bridges at the time of the scan. The unmatched fingerprints in the Tor metrics list may correspond to bridges we missed, offline bridges, or bridges configured to use a port other than 9001 or 443.

In response to other discovery attacks against Tor bridges [248], the Tor project has started to deploy obfsproxy [239], a wrapper that disguises client–bridge connections as random data in order to make discovery by censors more difficult. Obfsproxy nodes listen on randomized ports, which serves as a defense against discovery by comprehensive scanning.

2.4.5 Monitoring Service Availability

Active scanning can help identify Internet outages and disruptions to service availability without an administrative perspective. Previous studies have shown that active surveying (ICMP echo request scans) can help track Internet outages, but they have either scanned small subsets of the address space based on preconceived notions of where outages would occur or have performed random sampling [88, 131, 224]. High speed scanning allows scans to be performed at a high temporal resolution through sampling or comprehensively. Similarly, scanning can help service providers identify networks and physical regions that have lost access to their service.

In order to explore ZMap's potential for tracking service availability, we performed continuous scans of the IPv4 address space during Hurricane Sandy to track its impact on the East Coast of the United States. We show a snapshot of outages caused by the hurricane in Figure 2.8.

2.4.6 Privacy and Anonymous Communication

The advent of comprehensive high-speed scanning raises potential new privacy threats, such as the possibility of tracking user devices between IP addresses. For instance, a company could track home Internet users between dynamically assigned IP addresses based on the HTTPS certificate or SSH host key presented by many home routers and cable modems.

This would allow tracking companies to extend existing IP-based tracking beyond the length of DHCP leases.

In another scenario, it may be possible to track travelers. In 2006 Scholz et al. presented methods for fingerprinting SIP devices [223] and other protocols inadvertently expose unique identifiers such as cryptographic keys. Such features could be used to follow a specific mobile host across network locations. These unique fingerprints, paired with publicly available network data and commercial geolocation databases, could allow an attacker to infer relationships and travel patterns of a specific individual.

The ability to rapidly send a single packet to all IPv4 addresses could provide the basis for a system of anonymous communication. Rather than using the scanner to send probes, it could be used to broadcast a short encrypted message to every public IP address. In this scenario, it would be impossible to determine the desired destination host. If the sender is on a network that does not use ingress filtering, it could also spoof source addresses to obscure the sender's identity. This style of communication could be of particular interest to botnet operators, because it would allow infected hosts to remain dormant indefinitely while waiting for instructions, instead of periodically checking in with command and control infrastructure and potentially revealing their existence.

2.5 Scanning and Good Internet Citizenship

We worked with senior security researchers and our local network administrators to consider the ethical implications of high-speed Internet-wide scanning and to develop a series of guidelines to identify and reduce any risks. Such scanning involves interacting with an

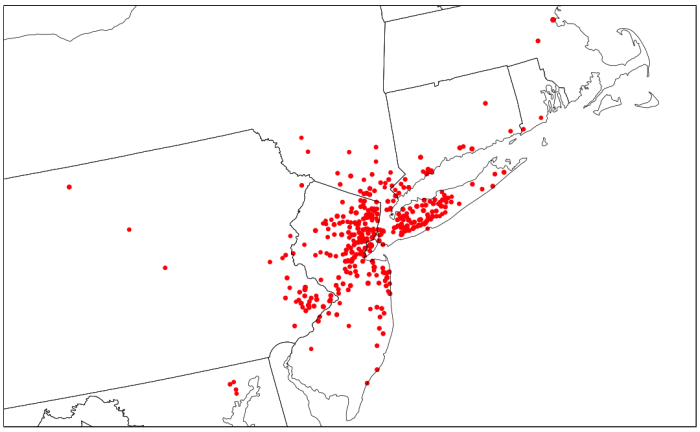


Figure 2.8: **Outages in the Wake of Hurricane Sandy** — We performed scans of port 443 across the entire IPv4 address space every 2 hours from October 29–31, 2012 to track the impact of Hurricane Sandy on the East Coast of the United States. Here, we show locations with more than a 30% decrease in the number of listening hosts.

enormous number of hosts and networks worldwide. It would be impossible to request permission in advance from the owners of all these systems, and there is no IP-level equivalent of the HTTP robots exclusion standard [152] to allow systems to signal that they desire not to be scanned. If we are to perform such scanning at all, the most we can do is try to minimize any potential for harm and give traffic recipients the ability to opt out of further probes.

High-speed scanning uses a large amount of bandwidth, so we need to ensure that our activities do not cause service degradation to the source or target networks. We confirmed with our local network administrators that our campus network and upstream provider had sufficient capacity for us to scan at gigabit speeds. To avoid overwhelming destination networks, we designed ZMap to scan addresses according to a random permutation. This spreads out traffic to any given destination network across the length of the scan. In a single probe TCP scan, an individual destination address receives one 40 byte SYN packet. If we scan at full gigabit speed, each /24 network block will receive a packet about every 10.6 seconds (3.8 bytes/s), each /16 network every 40 ms (1000 bytes/s), and each /8 network every 161 μs (250,000 bytes/s) for the 44 minute duration of the scan. These traffic volumes should be negligible for networks of these sizes.

Despite these precautions, there is a small but nonzero chance that any interaction with remote systems might cause operational problems. Moreover, users or network administrators who observe our scan traffic might be alarmed, in the mistaken belief that they are under attack. Many may be unable to recognize that their systems are not being uniquely targeted and that these scans are not malicious in nature, and might waste resources responding. Some owners of target systems may simply be annoyed and want our scans to cease. To minimize the risks from these scenarios, we took several steps to make it easy for traffic recipients to learn why they were receiving probes and to have their addresses excluded from scanning if so desired.

First, we configured our source addresses to present a simple website on port 80 that describes the nature and purpose of the scans The site explains that we are not targeting individual networks or attempting to obtain access to private systems, and it provides a contact email address to request exclusion from future scans. Second, we set reverse DNS records for our source addresses to "researchscan*x*.eecs.umich.edu" in order to signal that traffic from these hosts was part of an academic research study. Third, we coordinated with IT teams at our institution who might receive inquiries about our scan traffic.

For our ongoing Internet-wide HTTPS surveys (our largest-volume scanning effort), we took additional steps to further reduce the rate of false alarms from intrusion detection systems. Rather than scanning at full speed, we conducted each of these scans over a 12

1. Coordinate closely with local network admins to reduce risks and handle inquiries.
2. Verify that scans will not overwhelm the local network or upstream provider.
3. Signal the benign nature of the scans in web pages and DNS entries of the source addresses.
4. Clearly explain the purpose and scope of the scans in all communications.
5. Provide a simple means of opting out, and honor requests promptly.
6. Conduct scans no larger or more frequent than is necessary for research objectives.
7. Spread scan traffic over time or source addresses when feasible.

Table 2.5: **Recommended Practices** — We offer these suggestions for other researchers conducting fast Internet-wide scans as guidelines for good Internet citizenship.

hour period. We also configured ZMap to use a range of 64 source addresses and spread out probe traffic among them. We recognize that there is a difficult balance to strike here: we do not want to conceal our activities from system administrators who would want to know about them, but we also do not want to divert IT support resources that would otherwise be spent dealing with genuine attacks.

We provide a summary of the precautions we took in Table 2.5 as a starting point for future researchers performing Internet-wide scans. It should go without saying that scan practitioners should refrain from exploiting vulnerabilities or accessing protected resources, and should comply with any special legal requirements in their jurisdictions.

2.5.1 User Responses

We performed approximately 200 Internet-wide scans over the course of a year, following the practices described above. We received e-mail responses from 145 scan traffic recipients, which we classify in Table 2.6. In most cases, these responses were informative in nature, 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 scan source IP addresses, but we also received responses sent to our institution's help desk, our chief information security officer, and our departmental administrator.

We responded to each inquiry with information about the purpose of our scans, and we immediately excluded the sender's network from future scans upon request. In all, we excluded networks belonging to 91 organizations or individuals, totaling 3,753,899 addresses (0.11% of the public IPv4 address space). About 49% of the blacklisted addresses resulted from requests from two Internet service providers. We received 15 actively hostile responses that threatened to retaliate against our institution legally or to conduct a denial-of-service (DOS) attack against our network. In two cases, we received retaliatory DOS traffic, which was blacklisted by our upstream provider.

Small/Medium Business	41
Home User	38
Other Corporation	17
Academic Institution	22
Government/Military	15
Internet Service Provider	2
Unknown	10
Total Entities	145

Table 2.6: **Responses by Entity Type** — We classify the responses and complaints we received about our ongoing scans based on the type of entity that responded.

2.6 Related Work

Many network scanning tools have been developed, the vast majority of which have been optimized to scan small network segments. The most popular and well respected is Nmap ("Network Mapper") [168], a versatile, multipurpose tool that supports a wide variety of probing techniques. Unlike Nmap, ZMap is specifically designed for Internet-wide scanning, and it achieves much higher performance in this application.

Leonard and Loguinov introduced IRLscanner, an Internet–scale scanner with the demonstrated ability to probe the advertised IPv4 address space in approximately 24 hours, ultimately scanning at 24,421 packets per second [163]. IRLscanner is able to perform scanning at this rate by utilizing a custom Windows network driver, IRLstack [228]. However, IRLscanner does not process responses, requires a custom network driver and a complete routing table for each scan, and was never released to the research community. In comparison, we developed ZMap as a self-contained network scanner that requires no custom drivers, and we are releasing it to the community under an open source license. We find that ZMap can scan at 1.37 million packets per second, 56 times faster than IRLScanner was shown to operate.

Previous work has developed methods for sending and receiving packets at fast network line speeds, including PF_RING [86], PacketShader [128], and netmap [219], all of which replace parts of the Linux kernel network stack. However, as discussed in Section 2.3.1, we find that the Linux kernel is capable of sending probe packets at gigabit Ethernet line speed without modification. In addition, libpcap is capable of processing responses without dropping packets as only a small number of hosts respond to probes. The bottlenecks in current tools are in the scan methodology rather than the network stack.

Many projects have performed Internet-scale network surveys (e.g., [102, 130, 133, 137, 185, 208]), but this has typically required heroic effort on the part of the researchers. In

2008, Heidemann et al. presented an Internet census in which they attempted to determine IPv4 address utilization by sending ICMP packets to allocated IP addresses; their scan of the IPv4 address space took approximately three months to complete and claimed to be the first Internet-wide survey since 1982 [130]. Two other recent works were motivated by studying the security of HTTPS. In 2010, the Electronic Frontier Foundation (EFF) performed a scan of the public IPv4 address space using Nmap [168] to find hosts with port 443 (HTTPS) open as part of their SSL Observatory Project [102]; their scans were performed on three Linux servers and took approximately three months to complete. Heninger et al. performed a scan of the IPv4 address space on port 443 (HTTPS) in 2011 and on port 22 (SSH) in 2012 as part of a study on weak cryptographic keys [133]. The researchers were able to perform a complete scan in 25 hours by concurrently performing scans from 25 Amazon EC2 instances at a cost of around $300. We show that ZMap could be used to collect the same data much faster and at far lower cost.

Most recently, an anonymous group performed an illegal "Internet Census" in 2012, using the self-named Carna Botnet. This botnet used default passwords to log into thousands of telnet devices. After logging in, the botnet scanned for additional vulnerable telnet devices and performed several scans over the IPv4 space, comprising over 600 TCP ports and 100 UDP ports over a 3-month period [33]. With this distributed architecture, the authors claim to have been able to perform a single-port scan survey over the IPv4 space in about an hour. ZMap can achieve similar performance without making use of stolen resources.

2.7 Future Work

While we have demonstrated that efficiently scanning the IPv4 address space at gigabit line speeds is possible, there remain several open questions related to performing network surveys over other protocols and at higher speeds.

Scanning IPv6 While ZMap is capable of rapidly scanning the IPv4 address space, brute-force scanning methods will not suffice in the IPv6 address space, which is far too large to be fully enumerated [70]. This places current researchers in a window of opportunity to take advantage of fast Internet-wide scanning methodologies before IPv6-only services become common place. New methodologies will need to be developed specifically for performing surveys of the IPv6 address space.

Server Name Indication Server Name Indication (SNI) is a TLS protocol extension that allows a server to present multiple certificates on the same IP address [55]. SNI has not yet been widely deployed, primarily because Internet Explorer does not support it on Windows

XP hosts [159]. However, its inevitable growth will make scanning HTTPS sites more complicated, since simply enumerating the address space will miss certificates that are only presented with the correct SNI hostname.

Scanning Exclusion Standards If Internet-wide scanning becomes more widespread, it will become increasingly burdensome for system operators who do not want to receive such probe traffic to manually opt out from all benign sources. Further work is needed to standardize an exclusion signaling mechanism, akin to HTTP's robots.txt [152]. For example, a host could use a combination of protocol flags to send a "do-not-scan" signal, perhaps by responding to unwanted SYNs with the SYN and RST flags, or a specific TCP option set.

2.8 Conclusion

We are living in a unique period in the history of the Internet: typical office networks are becoming fast enough to exhaustively scan the IPv4 address space, yet IPv6 (with its much larger address space) has not yet been widely deployed. To help researchers make the most of this window of opportunity, we developed ZMap, a network scanner specifically architected for performing fast, comprehensive Internet-wide surveys.

We experimentally showed that ZMap is capable of scanning the public IPv4 address space on a single port in under 45 minutes, at 97% of the theoretical maximum speed for gigabit Ethernet and with an estimated 98% coverage of publicly available hosts. We explored the security applications of high speed scanning, including the ability to track protocol adoption at Internet scale and to gain timely insight into opaque distributed systems such as the certificate authority ecosystem. We further showed that high-speed scanning also provides new attack vectors that we must consider when defending systems, including the ability to uncover hidden services, the potential to track users between IP addresses, and the risk of infection of vulnerable hosts en masse within minutes of a vulnerability's discovery.

We hope ZMap will elevate Internet-wide scanning from an expensive and time-consuming endeavor to a routine methodology for future security research. As Internet-wide scanning is conducted more routinely, practitioners must ensure that they act as good Internet citizens by minimizing risks to networks and hosts and being responsive to inquiries from traffic recipients. We offer the recommendations we developed while performing our own scans as a starting point for further conversations about good scanning practice.

CHAPTER 3

Facilitating Easy Access to Scan Data

3.1 Introduction

Fast Internet-wide scanning has opened new avenues for empirically-driven security research, as evidenced by the recent surge in publications based on the technique (e.g., [24, 51, 58, 59, 63, 68, 78, 83, 99, 100, 122, 133, 153, 155, 165, 169, 253]). Yet while tools such as ZMap have reduced the time required to conduct large-scale port scans, collecting meaningful data through Internet-wide scanning has remained a specialized and labor-intensive process. Answering simple questions, such as "What fraction of HTTPS servers prefer forward-secret key exchange methods?", can take weeks of implementation and debugging, reducing the time security researchers have to focus on more important questions. In this specific case, the researcher would need to develop a high-performance application scanner to make HTTPS connections to hosts listening on port 443, test and fix problems with hosts that do not fully follow the TLS specification, run the actual scan, and then process many gigabytes of resulting data.

Before beginning this process, security researchers must negotiate with their institution's legal and networking teams for permission to conduct the scan, coordinate with their upstream network providers, and later respond to resulting abuse complaints. Many institutions (and independent researchers) lack the network facilities or administrative backing to perform scans. For these reasons, Internet-wide scanning has remained the province of a small number of research groups, which severely limits the applications to which this powerful methodology is applied.

In order to democratize Internet-wide scanning and enable researchers to efficiently ask questions about how security protocols have been deployed in practice, we have developed Censys, a cloud-based service that not only maintains an up-to-date snapshot of the hosts and services running across the public IPv4 address space, but also exposes this data through a search engine and API. In contrast to existing scanning tools, which have primarily focused on performing host discovery, Censys immediately produces results based on

full protocol handshakes, facilitates a community-driven approach to characterizing the exploding number of embedded devices and vulnerabilities on the Internet, and requires little or no user preparation.

To approximate a real-time "bird's eye view" of the Internet, Censys continually scans the public address space across a range of important ports and protocols. It validates this data and performs application-layer handshakes using a pluggable scanner framework, which dissects handshakes to produce structured data about each host and protocol. The resulting data is post-processed with an extensible annotation framework that enables researchers to programmatically define additional attributes that identify device models and tag security-relevant properties of each host. We operate Censys transparently and expose data back to the research community. In turn, we encourage external researchers to contribute both *application scanners* (to scan additional protocols) and *annotations* (to identify devices or properties) to Censys. In this way, Censys automates and centralizes the mechanical aspects of scanning.

Censys exposes data to researchers through a public search engine, REST API, publicly accessible tables on Google BigQuery, and downloadable datasets. The search interface enables researchers to perform full-text searches and query any of the structured fields and tags produced during scanning and post processing (e.g., `443.https.cipher_suite`). It supports full-text searches, regular expressions, and numeric ranges, and queries can be combined with Boolean logic. These queries can be run against a current snapshot of publicly accessible IPv4 hosts, Alexa Top 1 Million websites, and known X.509 certificates. After running a query, users can interactively explore the hosts, sites, and certificates that match their query, as well as generate statistical reports suitable for direct use in research.

As a simple example, Censys can identify the set of hosts in the U.S. that are currently vulnerable to Heartbleed with the query, `443.https.heartbleed.vulnerable: true and location.country_code: US`. From there, Censys can output a complete list of matching IP addresses and graph the distribution of the most common vulnerable device models. These queries complete in under one second.

To facilitate more complex analysis, we publish raw application handshakes and daily point-in-time snapshots of the structured data. These can be queried using SQL through publicly accessible Google BigQuery tables or downloaded in JSON form. Censys additionally exposes data through a public REST API that allows researchers to export raw query results, fetch statistical data, and view the historical state of specific hosts and networks.

We present Censys's data collection architecture in Section 3.3, explain how Censys presents data to researchers in Section 3.4, and describe our deployment in Section 3.5. We then illustrate Censys's potential in Section 3.6 by showing how it can be applied to easily

answer a range of questions from recent security studies, including measuring the impact of POODLE and tracking vulnerable industrial control systems.

Internet-wide scanning has already shown great potential for uncovering security problems and understanding the security of complex distributed systems. By moving scanning to the cloud, Censys dramatically reduces the effort needed to investigate these questions, enabling researchers to focus on asking more important questions rather than on the mechanics of answering them. Further, Censys allows the security community to increase global protocol coverage and provides a tractable solution for understanding the increasing number of embedded devices on the Internet. Simultaneously, it minimizes redundant scanning by research groups and minimizes the incoming network traffic monitored by network operators.

Censys is available free to the public at https://censys.io.

3.2 Good Internet Citizenship

As with any research conducted through active network probing, our work raises important ethical considerations. We carefully considered the impact of our experimental measurements and disclosure of our results. When reasoning about our impact, we considered a variety of stakeholders, from our local institution to Internet service providers and the owners of the remote systems. Although the community has yet to derive robust ethical standards for active measurement, our reasoning was guided by broad ethical principles, such as those embodied in the Menlo Report [38], as well as by the guidelines for ethical scanning set forth in the original ZMap work [101].

We coordinated with network administrators and IT leadership at our department, college, and institution, as well as with our upstream ISP, to ensure that our scans do not adversely impact network operations and that all support centers can route external inquiries to our team. Second, we signaled the benign intent of our activities. All of the scanning hosts have WHOIS records and reverse DNS entries that describe the intent of the scanning. Further, each scanning host runs a simple website on port 80 that describes the goals of the research, including what data we collect, and how to contact us. Third, we invite user exclusion requests and respond to requests within 24 hours. Fourth, all scans perform standards-compliant handshakes; we do not send malformed packets or handshakes.

Disclosure of scan data also raises ethical questions, since it exposes information about potentially vulnerable systems. To minimize harms, we deliberately choose to collect and distribute data that is, at least in principle, already publicly visible. Our scanners do not perform login attempts, deploy any exploits, or try to access non-public resource paths. Furthermore, we treat opt-out requests for scanning as a request to be removed from the

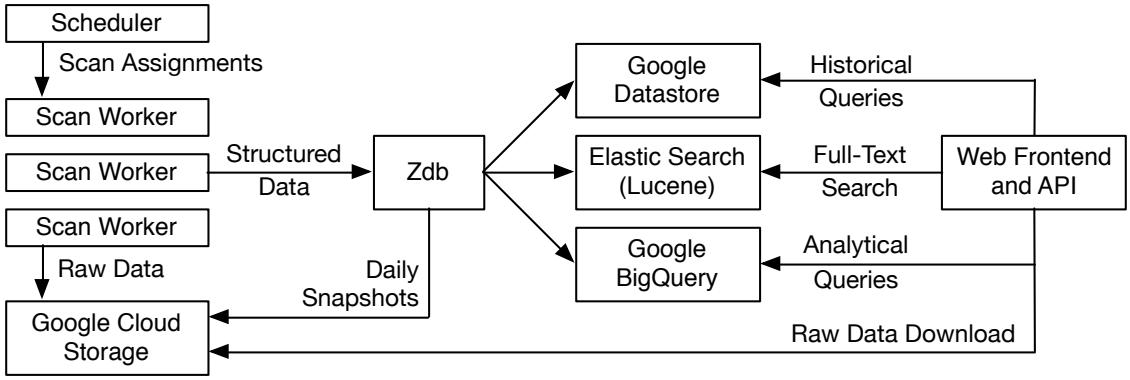


Figure 3.1: **Censys System Architecture** — Censys is driven by application scans of the IPv4 address space, which are scheduled onto a pool of scan workers. These workers complete scans, extract valuable fields, and annotate records with additional metadata in order to generate structured data about each host. These records are centrally managed in a custom database engine, ZDb, which maintains the current state of every host. ZDb feeds updated records to a web front-end where researchers can query the data.

search index, which allows remote administrators to decide whether or not to be included in Censys's public interface. Many network operators, after understanding the goals of our measurement work, have responded supportively and invited us to continue scanning them. Finally, it is our hope that by publishing scan data, carefully acquired and properly curated, we can reduce the need for Internet scanning performed by other researchers, and thus reduce the overall burden on destination networks.

In contrast, it is well established that attackers already use Internet-wide scanning to find vulnerable machines from botnets and bullet-proof hosting providers [98]. Thus, systems that are configured to expose data publicly are already at risk. Censys helps level the playing field by enabling legitimate researchers to study and enhance the security of these hosts by providing a source of reliable and ethically collected data.

3.3 Collecting Data

The data that powers Censys is collected through horizontal application scans of the public IPv4 address space, which are scheduled across a pool of scan workers. In the first step, we perform host discovery scans using ZMap [101], complete application handshakes with responsive hosts using pluggable application scanners, and derive structured fields (e.g., certificate subject or TLS cipher suite) from the handshake. We save and publish the raw handshakes, but continue further processing, validating the collected scan data, extracting valuable fields and annotating handshakes with additional metadata, such as device model and software version using user-defined annotations.

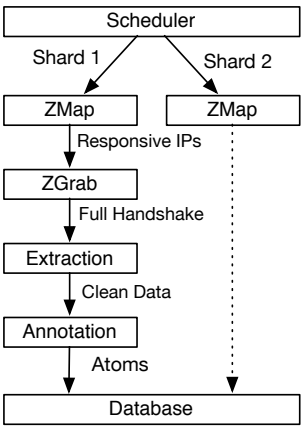


Figure 3.2: **Protocol Scanning and Annotation** — Each scan worker uses ZMap to perform host discovery for a shard of the IPv4 address, and completes protocol handshakes using pluggable application scanners. Censys extracts fields of interest and annotates records with additional metadata. The information from a protocol handshake is converted to an *atom*—a deterministic data structure describing a specific protocol on a host.

The structured, annotated data is then streamed to a central database, *ZDb*, which aggregates the horizontal scan results, pivoting the data and updating comprehensive records describing individual IPv4 hosts, Alexa Top 1 Million websites, as well as maintaining auxiliary collections of all found X.509 certificates and public keys. ZDb streams changes to downstream services and produces publishable point-in-time snapshots of hosts and websites, as well as differential updates to its collection of certificates and public keys.

There are several observations that led to this architecture. First, while horizontal scans measure a single aspect of a service (e.g., whether an HTTPS server supports SSLv3), research questions frequently depend on multiple scans. For example, calculating the percentage of HTTPS servers that support SSLv3 requires a generic TLS scan and an SSLv3 scan. Similarly, a device model may only be identifiable based on its HTTP page, but this information is useful when studying any protocol. Therefore, despite being collected by protocol, data should be grouped by host. Second, our framework needs to be extensible, and facilitate community involvement. Much of ZMap's success is due to user contributed probe modules, and we believe the same will be true for Censys. This is particularly true for annotating hosts and services given the exploding number of embedded devices on the Internet. In turn, Censys needs to operate transparently and provide data back to the community. Third, the number of scans and annotations will both grow over time; our architecture should scale linearly to handle this increased load.

3.3.1 Internet-Wide Scanning

In the first step of data collection, we use ZMap [101] to perform single-packet host discovery scans against the IPv4 address space. The hosts found by ZMap seed pluggable application scanners, which perform a follow-up application-layer handshake and produce structured JSON data describing a certain aspect of how a host is configured. Typically, application scanners only perform a single handshake and measure one aspect of how a service is configured. For example, we perform separate horizontal scans and use different pluggable scanners to measure how HTTPS hosts respond to a typical TLS handshake, whether hosts support SSLv3, and whether a host is vulnerable to the Heartbleed attack.

3.3.1.1 Pluggable Scanners

While we can use ZMap to perform host discovery for many protocols, every application scanner requires protocol specific code and Censys's long-term success is dependent on easily adding new protocols. In order to reduce the effort required to scan new protocols, Censys handles the details of a scan and expects a minimally featured application scanner. Specifically, Censys expects a self-contained Linux executable that performs application-layer handshakes with IP addresses input on `stdin` and produces structured JSON output that describes how the protocol is configured on `stdout`. Censys controls network bandwidth by rate limiting the IP addresses provided to scanners, splits scans across multiple scan workers using ZMap's built-in sharding [25], and guarantees that application scanners do not scan networks that have requested exclusion using ZMap's blacklist.

In order to protect our infrastructure, we require that application scanners operate without root privileges or kernel modifications. Lastly, we encourage researchers to output metadata about the scan in JSON form and to log errors in a standard format, which enables Censys to confirm whether a scan completed successfully. We hope that by requiring minimal feature-set and allowing flexibility in language, we not only reduce the effort required for our team to add additional protocols, but also encourage an external community that develops new scanners.

3.3.1.2 Scheduling Scans

While it would be technically simplest to measure every aspect of a protocol at once, this frequently involves making multiple handshakes, which could potentially inundate a host. Instead, we choose to perform scans independently, relying on our data processing pipeline to aggregate the data from different scans to reduce the load on individual hosts.

Internally, scans are referenced by the tuple (port, protocol, subprotocol, network destination), e.g., (443, https, heartbleed, ipv4_shard1), and individual scan executions are

referenced by scan and timestamp. We currently maintain a master scan schedule; we plan to automatically schedule scans moving forward to better distribute network load.

3.3.2 ZGrab: Our Application Scanner

We are releasing a fast and extensible application scanner, ZGrab, which meets the previous specifications and facilitates the rapid development of new types of scans. At this time, ZGrab supports application handshakes for HTTP, HTTP Proxy, HTTPS, SMTP(S), IMAP(S), POP3(S), FTP, CWMP, SSH, and Modbus, as well as StartTLS, Heartbleed, SSLv3, and specific cipher suite checks. On a dual-Xeon E5-2640 (6-cores at 2.5 GHz) system with an Intel X520 ethernet adapter, ZGrab can complete HTTPS handshakes with the full IPv4 address space in 6h20m, and a banner grab and StartTLS connection with all publicly accessible SMTP hosts in 3h9m, 1.86k and 1.32k hosts/second respectively.

ZGrab is implemented in Go, which we chose based on its native concurrency support [203], safety compared to other low-level languages, and its native cryptographic libraries [26]. The framework allows scanners to be defined as a serial chain of network events. By default, ZGrab will only perform one event per host, connect, which simply opens a TCP connection. Events can be as simple as reading or writing data, or more advanced, such as initiating a TLS handshake. For example, the HTTPS scanner is implemented as a connection event and a TLS handshake event. To extend ZGrab to support scanning for StartTLS support among SMTP servers, we added events to read the SMTP banner, write the SMTP EHLO command, read the SMTP response, send the StartTLS command, read the response, and perform the TLS handshake: a total 62 LoC. The ZGrab framework handles concurrent connections, as well as logging and generating JSON documents that describe the connection.

All of the protocols in the initial Censys deployment use ZGrab and we encourage other researchers to consider using it as a starting point for developing other application scanners. We are releasing and maintaining ZGrab as a standalone open-source tool as part of the ZMap Project[1]. ZGrab can be used independently of Censys and works in conjunction with ZMap: ZMap quickly identifies hosts and ZGrab produces structured data about each of those hosts.

3.3.3 Validation, Extraction, and Annotation

The raw JSON data produced by application scanners is collected by Censys, where it is validated, transformed into a structured schema, and annotated with additional metadata (e.g., device manufacturer and model), before being streamed into our central database.

[1] ZGrab is available at https://github.com/zmap/zgrab.

3.3.3.1 Validation

Censys validates scan data in two ways. First, we extended ZMap to detect variances in network responses during the host discovery stage. If the scan response rate falls below a set threshold at any time, varies more than a set amount during the scan, reaches a maximum number of sendto failures, or if *libpcap* is unable to keep up and drops a set number of packets, the scan automatically terminates and is rescheduled. Second, Censys validates a scan at its completion and rejects that scans where ZMap's or the application scanner's response rates fall outside of a static bound or deviate more than 10% from the median of the scans that completed over the last two weeks; rejected scans are manually inspected afterwards. These checks are primarily in place in order to detect transient network failures, human error in configuration, and coding errors.

3.3.3.2 Extraction

Application scanners output raw data about every aspect of an application handshake in a format analogous with the network handshake. For example, in the case of TLS, client and server randoms are output as part of the Client and Server Hello messages. While this data is needed for some research, many of these fields are not helpful when searching for hosts or identifying devices, and would cause unnecessary churn in our database. Similarly, commonly searched fields are nested deep within network protocol messages, making them hard to find. We save and publish the raw application scanner output, but then extract significant values and transform handshake data into consistent, structured records that conform to a published schema. We further output records in a deterministic manner during this process (i.e., the record has the same cryptographic hash if no configuration changes have occurred), which allows us to reduce load later by discarding records that contain no changes. We refer to these deterministic records that represent how a service is configured as *atoms*.

3.3.3.3 Annotation

While the output from application scanners can be used to identify a device model or version, these details are not directly exposed by a scanner. Instead, they frequently require a small amount of logic (e.g., running a regular expression against the HTTP server header or certificate subject). To facilitate adding this type of metadata, Censys allows researchers to define annotations—small functions—that can inject additional metadata fields (e.g., `device_module`) or attach simple tags (e.g., `IPMI` for server management cards) to hosts, websites, and certificates. Annotations are defined as standalone Python functions that are provided read-only access to the structured data that Censys generates from each scan. We show an example annotation for labeling Dell iDRAC remote management cards in

```
@tag(port=443, proto="https", subproto="tls")
def dell_idrac(d):
  subject = d.443.https.certificate.subject
  if subject.ou == "Remote Access Group" and subject.org == "Dell Inc.":
    return {"hw_manufacturer": "Dell Inc.", "hw_model": "iDRAC",
        "tags":["IPMI",]}
```

Figure 3.3: **Dell iDRAC Annotation** — Censys supports community maintained annotations—simple Python functions—that append additional metadata and tags to records. Here, we show the tag for Dell iDRAC remote management cards.

Figure 3.3.

We encourage researchers (and end-users alike) to contribute annotations for new types of devices and vulnerabilities. We are hosting our repository of annotations, along with our transformations and schemas as a standalone open source project, ZTag, on GitHub (http://github.com/zmap/ztag). We note that when ZTag is paired with ZMap and ZGrab, researchers can independently reproduce the entire data processing pipeline that Censys uses and independently generate the same data (Figure 3.2).

Port	Protocol	SubProtocol	Port Open (Hosts)	Full Handshake (Hosts)	Raw Record Size (KB)	Processed Size (KB)	% Diff 1 Day	Database Size on Disk
80	HTTP	GET /	77.3 M	66.8 M	.69 (1.8)	.32 (.096)	11.5%	10.9 GB
443	HTTPS	TLS	47.1 M	33.3 M	3.7 (4.9)	4.5 (1.4)	8.5%	50.1 GB
443	HTTPS	SSLv3	43.1 M	22.5 M	2.8 (3.9)	.08 (.0001)	6.8%	1.5 GB
443	HTTPS	Heartbleed	47.1 M	33.1 M	3.6 (4.8)	2.3 (.002)	4.4%	4.8 GB
7547	CWMP	GET /	55.1 M	44.3 M	.3 (.3)	.34 (.09)	28.1%	6.5 GB
502	MODBUS	Device ID	2.0 M	32 K	.19 (.20)	.10 (.08)	10.6%	0.0 GB
21	FTP	Banner Grab	22.9 M	14.9 M	.08 (.09)	.33 (.31)	7.5%	9.0 GB
143	IMAP	Banner Grab	7.9 M	4.9 M	2.8 (4.1)	2.2 (8.9)	3.3%	7.0 GB
993	IMAPS	Banner Grab	6.9 M	4.3 M	6.6 (4.2)	4.9 (8.4)	2.0%	11.5 GB
110	POP3	Banner Grab	8.8 M	4.1 M	2.5 (3.9)	2.3 (.44)	4.4%	6.9 GB
995	POP3S	Banner Grab	6.6 M	4.0 M	6.4 (4.1)	2.4 (.4)	1.9%	6.9 GB
25	SMTP	Banner Grab	14.7 M	9.0 M	1.9 (3.6)	1.5 (1.2)	5.8%	8.9 GB
22	SSH	RSA	14.3 M	14.3 M	1.0 (.2)	.6 (.2)	13.8%	5.8 GB
53	DNS	OpenResolver	12.4 M	8.4 M	.05 (.001)	.145 (0)	29.8%	0.7 GB
123	NTP	Get Time	1.6 M	1.2 M	.02 (.0006)	.145 (0)	92.8%	0.1 GB
1900	UPnP	Discovery	9.5 M	9.5 M	.051 (.002)	.10 (0.0)	37.2%	0.6 GB

Table 3.1: **Scanned Protocols** — We scan 16 protocols in our initial implementation. For each protocol and subprotocol we scan, we show the average size and standard deviation for raw and transformed records, as well as the percent-change in records across two days of scans. Most protocols have a less than a 15% turnover rate between consecutive days.

3.3.4 Challenges in Aggregating Data

Scan workers act independently and statelessly; an individual scan worker is not aware of other scan workers nor does it have any prior knowledge of a host's state. Therefore, a worker does not know whether a scanned host has changed or moved since a previous scan. Instead, workers stream all structured data to a central database to be processed. This vastly simplifies the development of application scanners and facilitates linearly increasing computational power by adding additional scan workers. However, this abstraction requires a more complex data processing pipeline that can process the incoming stream of data. For just the five largest protocols in our initial deployment (Table 3.1), this amounts to processing at least 330m records per day—a sustained 3.8k writes/second.

Our database needs are theoretically simple: we need to (1) parse incoming records and update the relevant part of each host record, maintain the current state of hosts, (2) stream changes to downstream, user-facing, services, (3) efficiently dump the database to JSON to produce daily snapshots. At a small scale, this could be easily handled out-of-the-box by one of many NoSQL database engines. However, we find that popular NoSQL engines perform updates prohibitively slowly given our workload (Table 3.2).

Database	New Records (rec/sec)	No Differences (rec/sec)	Consecutive Day (rec/sec)	Size on Disk
ZDb	58,340	136,664	110,678	1.60 GB
MongoDB	1,059	1,441	1,392	13.67 GB
Cassandra	501	511	506	3.40 GB

Table 3.2: **NoSQL Engine Comparison** — We compare ZDb against the two leading NoSQL engines [229], MongoDB and Apache Cassandra by loading a full scan of FTP. We find that ZDb is 80× faster than MongoDB and 219× faster than Cassandra when updating a consecutive day. For context, a single HTTP scan produces 2.4 K records/second.

We tested the two most popular NoSQL engines [229], MongoDB 2.6.7 and Apache Cassandra 2.1.2, under three scenarios: (1) importing a new protocol for the first time, (2) re-importing the same dataset, and (3) loading two consecutive days of scans. These tests approximate the worst case, best case, and typical daily use case for Censys. We specifically loaded a banner grab scan of FTP (one of our simplest protocols) from February 12 and 13, 2015, which contained an average 14.5m records. Apache Cassandra consistently updated at about 500 records/second for all cases. MongoDB updated at an average 1,400 records/second when updating between two consecutive days, but consistently slowed as the database grew. At these rates, Cassandra would require 37 hours to update a single HTTP scan on our server; MongoDB would require 13 hours. MongoDB and Cassandra further required

8.5× and 2.1× the disk space of the raw data. We performed these experiments on an Intel branded server with dual Xeon E5-2640 processors (12 cores at 2.50 GHz), 128 GB of DDR3 memory, and a Samsung 850 Pro 1 TB SSD drive. We ran MongoDB with `w=0` write concern, which provides acknowledgment from the server that the request was received and could be processed, but not that it has been flushed to disk.

While it is possible to horizontally scale both database engines across a large number of servers, we observe that our needs differ greatly from a typical OLTP database and could likely be solved with a simple, custom database. Our first observation is that the majority of updates contain unchanged data (91.5% in the case of HTTPS; 88.5% for HTTP) and can be safely discarded. While MongoDB and Cassandra did not handle unchanged records significantly faster than actual updates, we could quickly discard these records. Second, if changed data is immediately streamed to downstream user-facing services, we do not need to quickly serve arbitrary reads. Instead, reads will only be necessary as part of large batch jobs. Therefore, we do not need to cache user data in memory in order to support fast queries. Instead, we should focus on optimizing for quickly processing incoming records and organizing the data to facilitate efficient batch jobs. Last, updates are streamed from scan workers, which are architected to be linearly scaled. If there are expensive operations to be performed on every record, these can be offloaded to the database client in order to reduce load on the central database.

With these considerations in mind we developed ZDb, which aggregates the records generated by scan workers, maintains the current state of hosts on the IPv4 address space, websites in the Alexa Top 1 Million Sites, and curates auxiliary collections of all X.509 certificates and public keys we've encountered. ZDb is able to process upwards of 110K records/second for a typical workload—a 219× speedup over Cassandra and 80× speedup over MongoDB. We describe ZDb's architecture and our data processing pipeline in the next section.

3.3.5 Censys Data Flow

After a scan worker finishes processing a host, it serializes the annotated, structured data into a Google Protocol Buffer message [116], which it sends to the central ZDb server along with the SHA-1 fingerprint of the message and a key describing what was scanned. These messages are queued in memory and processed by a pool of worker threads, which deserialize and validate the outer record, and check whether the record has changed since the last scan (using the attached SHA-1 fingerprint). If the record has changed, the new record is written to disk, and enqueued in external Redis queues for downstream services (e.g., the database of historical records, the search index, and other institutions subscribed to

a live data feed). If the record has not changed since the latest scan, we simply mark the record as having been seen in the most recent scan. When the scan completes, we prune any records that were not updated or marked as having been seen in the scan. Analogous to the IPv4 database, we maintain a collection of records for the Alexa Top 1 Million domains, as well as auxiliary collections of all seen X.509 certificates and public keys.

Internally, data is stored on disk using RocksDB [105], an embeddable key-value store optimized for flash storage. RocksDB buffers updates to a small in-memory table and on-disk journal, and, in another thread, flushes changes to a log-structured merge-tree on disk. The records stored in RocksDB consist of the serialized protobuf messages generated by the scan workers. We note that because data is written to disk as a log-structured merge-tree, we maintain a global ordering of all records, which we use to logically group multiple records that describe a single host. Similarly, records describing a single network are grouped together. This allows us to efficiently generate daily snapshots of the IPv4 address space by performing a single, linear pass of the database, grouping together all records describing a single host, and outputting a structured JSON document describing all measured aspects of each host.

All of the functionality in ZDb could be achieved using only RocksDB, but would require a disk read to process every incoming record. To improve performance, we cache the SHA-1 fingerprints of current records, along with whether the record were seen in the latest scan using an in-memory Judy Array. With this additional optimization, we no longer need to make random reads from RocksDB during processing and can process all incoming records that contain no changes without touching disk. We then update when each record was seen at the end of the scan during the prune process, which already performs a linear pass of the records on disk. With these optimizations, ZDb is able to process 58k records per second in the worst case, 137k records/second in the best case, and 111k records/second in the daily workload using the same metrics we used to measure MongoDB and Apache Cassandra.

We would be remiss not to mention that because records are queued in memory before they are processed, and because we cache which records have been seen in memory until a scan finishes, ZDb would lose the data associated with a particular scan if the server crashed (e.g., due to a kernel panic). While this is non-optimal, we find this risk acceptable, because a fresh scan can be completed in a matter of hours, which would likely be similar to the amount of time needed to investigate the crash, recover the database, and finish the remainder of a scan. We take a similar approach to managing failures during a scan. If a scan worker crashes, or if scan validation fails for any reason, we start a new scan rather than try to recover the previous scan.

3.4 Exposing Data

To be successful, Censys needs to expose data back to the community, which ranges from researchers who need to quickly perform a simple query to those who want to perform in-depth analysis on raw data. In order to meet these disparate needs, we are exposing the data to researchers through several interfaces, which offer varying degrees of flexibility: (1) a web-based query and reporting interface, (2) a programmatic REST API, (3) public Google BigQuery tables, and (4) raw downloadable scan results. We further plan to publish pre-defined dashboards that are accessible to users outside of the research community. In this section, we describe each of these interfaces in depth.

3.4.1 Search Interface

The primary interface for Censys is a search engine that allows researchers to perform full-text searches and structured queries against the most recent data for IPv4 hosts, the Alexa Top 1 Million websites, and known certificates. For example, a researcher can find all hosts currently vulnerable to Heartbleed in the United States with the query: `443.https.heartbleed.vulnerable: True AND location.country_code: US`. This query executes in approximately 250 ms and users are presented with the hosts that meet the criteria, along with basic metadata, which includes the breakdown of the top ASes, countries, and tags. Users can view the details of any host, as well as generate statistical reports.

3.4.1.1 Search Syntax

The search interface supports basic predicate logic, ranges, wildcards, and regular expressions . Users can perform simple full-text searches as well as query any structured field generated during the scan process, including user annotations and system-maintained metadata (e.g., location and network topology).

3.4.1.2 Viewing Individual Records

Users can view the details of any host, certificate, or domain returned by a query. This includes a user-friendly view of how each service is configured, the most recent raw data describing the host, user-provided metadata and tags, and historical scan data. We similarly display geographic location, routing, and WHOIS information.

3.4.1.3 Dynamic Reports

Once a query completes, users can generate reports on the breakdown of any field present on the resulting datasets. For example, users can view the breakdown of server chosen cipher suites for IPv4 HTTPS hosts with browser-trusted certificates by performing the query 443.https.tls.validation.browser_trusted: True and generating a report on

Interface	Query	Time
Web	`80.http.get.headers.server:*`	218 ms
API	`25.smtp.banner.banner:gsmtp`	82 ms
API	`ip:1.2.3.4`	12 ms
Report	`443.https.tls.certificate.issuer_dn`	417 ms
Report	`502.modbus.device_id.product_name`	11 ms

Table 3.3: **Censys Response Times** — Censys can be used to search records, and to create aggregations over fields in the search results. Here, we show example searches and aggregations along with their execution times. All of the queries completed in under 500 ms.

443.https.cipher_suite.name.

3.4.1.4 Backend

The search interface and reports are powered by Elasticsearch [39], an open-source project that front-ends Apache Lucene [35]. We maintain three indexes within Elasticsearch: IPv4 hosts, Alexa Top 1 Million websites, and all known certificates; ZDb updates the three indexes in real time. All updates are also appended to a Google Cloud Datastore collection, which is used to serve the history of each host. Our web front-end is implemented in Python using the Pylons Pyramid Framework, and is hosted on Google App Engine. We plan to rate-limit the web interface, using session-based token buckets, in order to prevent screen scraping and encourage developers to use the REST API we describe in the next section for programmatic access. We present the response time for sample queries in Table 3.3.

3.4.2 Programmatic Access

Censys has a programmatic API that provides equivalent functionality as the search interface, but presents JSON results and follows the semantics of a REST API. For example, researchers can get the history of an IPv4 host by doing a GET request for https://censys.io/api/ipv4/8.8.8.8/history. To prevent abuse, we require users to use a key, but are happy to provide these to researchers.

3.4.3 SQL Interface

We recognize that not all research questions can be answered through the search interface we described. This is particularly true for historical queries, because we only expose the most recent data. To support more complex queries, we are exposing Google BigQuery tables that contain the daily ZDb snapshots of the IPv4 address space and Alexa Top 1 Million Domains, along with our auxiliary collection of certificates and public keys. Google BigQuery is a Dremel-backed cloud database engine designed for performing large analytical queries.

Queries require 10–20 seconds to execute, but allow a full SQL syntax and are not restricted to specific indexes. Authenticated researchers can perform queries through the Censys web interface, or access the tables directly using their own Google Cloud Accounts.

3.4.4 Raw Data

Lastly, we are publishing all of the raw data from our scans, along with our curated ZDb snapshots of the IPv4 address space, Alexa Top 1 Million websites, and known certificates. We will be posting these as structured JSON documents, along with data definitions, and schemas for common databases at censys.io/data. We previously posted scan data on https://scans.io, a generic scan data repository that our team hosts. We will continue to maintain the scans.io interface, provide continued access to our historical datasets, and allow researchers to upload other data. However, we will no longer post our regular scans to https://scans.io, but rather encourage users to download these directly from Censys's web interface.

3.4.5 Protocol Dashboards

While Censys's primary goal is to answer researchers' specific queries, the backend similarly supports the types of queries needed to generate pre-determined reports and dashboards. We plan to publish dashboards on Censys' website, which present various perspectives of how protocols are deployed in practice. At initial release, we will be releasing a *Global HTTPS Dashboard* that presents how well HTTPS has been deployed in practice, an *Alexa HTTPS Deployment Dashboard* that shows historical trends in HTTPS deployment and which high-ranking sites have not deployed HTTPS, and dashboards for each of the recent HTTPS vulnerabilities, which will supersede the *Heartbleed Bug Health Report*, *POODLE Attack and SSLv3 Deployment*, *Tracking the FREAK Attack*, and *Who is affected by Logjam?* sites. Initially, dashboards will be statically defined. However, we encourage researchers to contribute reports at the completion of research projects, and moving forward we hope to allow everyone to dynamically define new reports.

3.5 Initial Deployment

We are releasing Censys with the sixteen protocols listed in Table 3.1, which we are scanning from the University of Michigan.

3.5.1 Scanning

We originally scheduled all protocols to run on a daily basis, but quickly reduced scan speed and frequency due a non-negligible uptick in complaints and exclusion requests. Instead, we have switched to scheduling scans based on protocol turnover. We specifically

scan HTTP, HTTPS, and CWMP on a daily basis; SSH, Modbus, FTP, DNS, NTP, UPnP, SMTP, and SSH biweekly; and IMAP, IMAPS, POP3, POP3, POP3S, HTTPS/SSLv3, and HTTPS/Heartbleed weekly. All scans are performed over a 24 hour hour period. We plan to further fine tune this schedule based on which protocols are frequently searched and downloaded after public release, and will post updated schedules on the web interface.

3.5.2 Backend

During initial testing, when we scanned every protocol daily, we were able to consistently complete all sixteen scans from a pool of 12 scan workers and ZDb comfortably handled upwards of 40 full-speed scans on a single host—an Intel branded server with two Intel Xeon E5-2640 (6 cores at 2.50GHz) processors, 192 GB of DDR3 memory, and RAID 1+0 with four Intel 850 Pro 1 TB SSD drives. Our scan workers are Dell PowerEdge 1950s, with a Quad-Core Intel Xeon X5460 at 3.16 Ghz, 16 GB of memory, and a local 1 TB 7200 RPM SATA drive.

3.5.3 Frontend

Our front-end is powered by a number of components. The web interface itself is implemented as a Python Pyramid project, served through Google App Engine, our search backend is powered by Elasticsearch and Apache Lucene, historical data is stored in Google Datastore, and historical and advanced queries utilize Google BigQuery. With the exception of Elasticsearch, these services will autoscale based on load. During private testing, we were comfortably able to serve Elasticsearch requests for a small number of internal users from a single server (2 x Intel Xeon E5-2640 (6 cores at 2.50GHz) processors, 192 GB of DDR3 memory, and RAID 1+0 with four Intel 850 Pro 1 TB SSD drives). Our initial public Elasticsearch deployment runs on Google Compute Engine and consists of six backend data nodes, each with 52 GB of memory, 8 VCPUs, and 500 GB solid state storage, and two front-end nodes. We can add additional nodes to the cluster as load dictates.

3.6 Applications

Exposing scan data to new sets of researchers, who do not otherwise have access to scanning methodologies was one of the primary motivations for developing Censys. In this section, we show how Censys can be used to easily answer frequently asked Internet measurement questions, as well as questions from recent security studies.

3.6.1 Industrial Control Systems

SCADA (Supervisory control and data acquisition) systems provide a communication channel for computer systems to control industrial equipment, such as motors, generators,

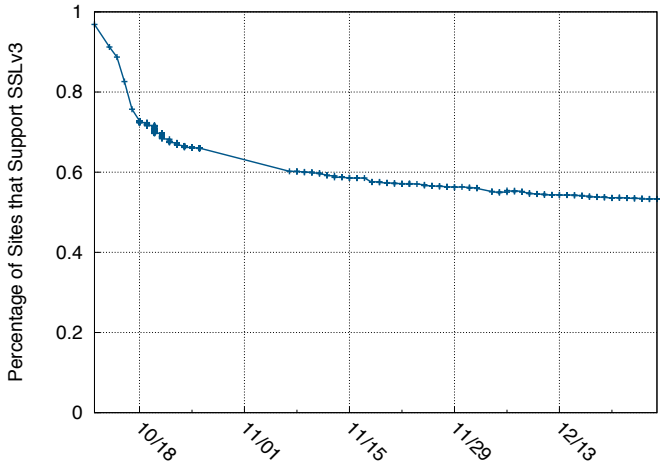


Figure 3.4: **SSLv3 Deprecation** — Censys tracks both the IPv4 and Alexa Top 1 Million websites. We track the deprecation of SSLv3 of both after the POODLE announcement using Censys. We find that the support for SSLv3 has dropped from 96.9% to 46.0% between October 2014 and February 2015 for the Top 1 Million Websites.

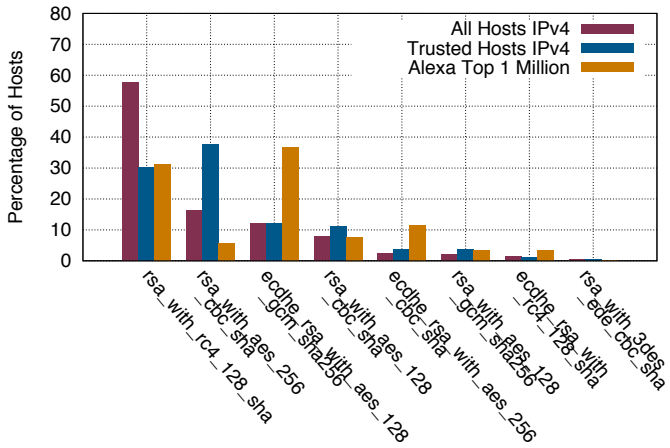


Figure 3.5: **HTTPS Cipher Suites** — We show the breakdown of cipher suites chosen by all IPv4 hosts, hosts with browser trusted certificates, and the Alexa top million domains using numbers returned by Censys's web interface.

Country	Modus Devices	
United States	4723	24.7%
Spain	1,448	7.58%
Italy	1,220	6.39%
France	1,149	6.02%
Turkey	884	4.63%
Canada	822	4.30%
Denmark	732	3.83%
Taiwan	682	3.57%
Europe	615	3.22%
Sweden	567	2.97%
Total	12,842	67.23%

Table 3.4: **Top Countries with Modbus Devices** — We identified Modbus hosts in 117 countries, with the top 10 countries accounting for 67% of the total costs, and nearly one-quarter of all Modbus hosts we identifed are located in the United States.

Device Type	Count
Modbus Ethernet Gateway	1,440
Programmable Logic Controller	1,054
Solar Panel Controller	635
Water Flow Controller	388
Power Monitor/Controller	158
Touchscreen System Controller	79
SCADA Processor/Controller	99
Environment/Temperature Sensor	10
Cinema Controller	5
Generic Modbus Device	28,750

Table 3.5: **Modbus Devices** — We used Censys to categorize publicly available industrial control systems that support the Modbus protocol.

Vulnerability	Alexa	IPv4	IPv4 Trusted
Heartbleed	1.16%	0.96%	0.19%
SSLv3 Support	46.0%	55.8%	34.7%
SSLv3 Only	0.05%	2.9%	0.07%

Table 3.6: **Heartbleed and SSLv3** — We show a breakdowns for the Heartbleed vulnerability and SSLv3 support for HTTPS hosts in the IPv4 address space and the Alexa Top 1 Million.

Expires	Count	%SHA-1	% Total	Chrome
2015	6.86 M	60.2%	46.0%	Secure
2016	2.84 M	25.0%	19.0%	Warning
2017+	1.69 M	14.8%	11.3%	Insecure

Table 3.7: **SHA-1 Prevalence** — Chrome is beginning to mark sites with SHA-1 signed certificates as insecure. We used Censys to measure the fraction of trusted sites with SHA-1 certificates and how they appear in Chrome.

and physical sensors. SCADA-enabled devices are used extensively in industrial and infrastructure-critical systems, ranging from factories and power plants to HVAC systems and water treatment facilities. Attacks on these systems are particularly dangerous because SCADA devices bridge the gap from virtual to physical and configuration changes can have devastating consequences. In one recent example, the compromise of a blast furnace control system resulted in "massive damage" to a German steel mill in 2014 [260].

One of the primary SCADA protocols, Modbus, was originally designed for local communication over a serial connection, but has since been extended to operate over networks and the Internet [182]. The protocol contains no authentication, and as a result, publicly accessible Modbus devices are an inherent security risk. To show how Censys can be used to characterize these devices, we implemented annotations to identify different types of Modbus devices. With 42 annotations, we categorized 92% of Modbus devices that responded to a device identification request. These annotations contain device classification, hardware manufacturer, software name, and version. We queried Censys for modbus devices and aggregated hosts based on type, which identified 32,622 hosts and completed in 21 ms. We show the breakdown of device types in Table 3.5.

By querying Censys, we find that Internet-connected Modbus devices are pervasive, despite the total lack of security in the protocol. In situations where remote access is required, good security practice dictates that Modbus devices should be access-controlled through the use of a firewall or VPN [182]. Unfortunately, Censys identified many of the publicly accessible Modbus devices as network-enabled SCADA processors and gateways, capable of forwarding instructions to and controlling other networks of SCADA devices which might not be directly connected to the Internet. Censys located devices spread over 1,880 ASes and 111 countries, with the top countries accounting for 77% devices (Table 3.4).

3.6.2 Heartbleed, Poodle, and SSLv3

The security community watched two catastrophic TLS vulnerabilities unfold in 2014: Heartbleed [74] and Poodle [183]. Heartbleed was caused by an implementation error in

OpenSSL that resulted in servers publicly leaking private data, including cryptographic keys and login credentials. Poodle was caused by fundamental flaw in the SSLv3 protocol, which allowed attackers to man-in-the-middle connections. Both of these vulnerabilities garnered both widespread research [94, 95, 100] and media attention. The fundamental question of "What hosts are still vulnerable?" surrounded the disclosures, and Internet-wide scanning was the key tool used for both understanding the impact of the vulnerabilities and facilitating Internet-wide notifications. Unfortunately, data describing the susceptibility of the Internet stagnated after initial publications. Despite this, the question of who is vulnerable remains important.

To determine what hosts with browser trusted certificates remained vulnerable to Heartbleed, we queried Censys for `443.https.certificate.signature.valid:true` and aggregated the `443.https.heartbleed_ vulnerable` field. Similarly, to determine what percentage of hosts only supported SSLv3 we queried for HTTPS hosts and aggregated the `443.https.tls_version.name` field, which completed in 229 ms. We provide breakdowns of the current state of Heartbleed and SSLv3 in Table 3.6.

Despite Heartbleed being disclosed over a year ago, over 1% of the Alexa Top 1M domains remain vulnerable. Additionally, 46% of HTTPS-enabled Alexa Top 1M sites still support SSLv3, down from 97% at the disclosure of POODLE four months ago. All of the data used to make these measurements about the current state of the Internet can be publicly queried on Censys's web interface.

We acknowledge that not all vulnerabilities can immediately be detected upon disclosure without some level of code modification. However, the data processing and application scanner framework in Censys allows researchers to quickly respond to vulnerabilities and to easily develop a custom scan module, if necessary. For example, in the case of Heartbleed, a minor modification to ZGrab to send a custom Heartbeat packet was all that was needed in order to stream Heartbleed scan data into Censys. Realistically, automated measurement in Censys can be started within a few hours of vulnerability disclosure. As the protocol coverage of Censys increases, we except the need for custom scan modules will further decrease. In the case of POODLE, FREAK, and Logjam, measurement of supported TLS versions and cipher suites would have been sufficient in detecting vulnerability trends immediately at the time of disclosure.

3.6.3 Institutional Attack Surface

Managing large, publicly accessible networks is an involved and complicated process. Censys can be used by organizations to measure their external-facing attack surface. Network-connected devices can be difficult to keep track of, and users may mistakenly open up

devices and services intended to be private. Censys supports queries based on network blocks and ASes, which an organization can use to easily export all of the data that Censys has gathered about their publicly-accessible services. This data can be used to identify mistakenly exposed or vulnerable devices, as well as identify devices that may have been overlooked when patching software.

Unfortunately, these mistakenly exposed devices might not only present a security risk to the institution hosting them, but also to the entire Internet. For example, misconfigured public NTP and DNS resolvers are the major cause of the new trend in amplification DDoS attacks [83, 155]. Amplification attacks can be globally prevented by eliminating publicly accessible open resolvers and NTP servers. As a result, several initiatives such as The Open Resolver Project [175] and The Open NTP Project [111] provide free scanning services on networks of up to 1,024 hosts for network administrators to use to identify misconfigured or publicly accessible devices that could be leveraged for amplification attacks. Censys removes the need for service-specific and vulnerability-specific scanning initiatives. Censys provides similar services as both of these initiatives, and is both real-time and Internet-wide.

3.6.4 Deprecating SHA-1

The Chrome Security team is spearheading an effort to deprecate HTTPS certificates signed using SHA-1, citing the decreasing cost of collision attacks [200]. Chrome now shows certificates signed with SHA-1 and expiring before 2016 as secure, displays a warning for those expiring between 2016 and 2017, and rejects SHA-1 signed certificates expiring in 2017 or later as insecure.

We used Censys to characterize the current prevalence of SHA-1 signed certificates for HTTPS hosts with browser trusted certificates. Specifically, we queried Censys to find all browser-trusted certificates expiring in 2015, 2016, and 2017 or later and used Censys's aggregation feature to bucket results by signature algorithm. We find that 76.3% of trusted IPv4 hosts currently use a SHA-1 signature and show the breakdown of SHA-1 certificates and their status in Chrome in Table 3.7.

3.6.5 Cipher Suites

The security of the TLS protocol fundamentally relies on the use of strong cipher suites. However, servers and clients often have to make a tradeoff between level of security and compatibility. When performing a TLS handshake, ZGrab offers the cipher suites implemented by the Golang TLS library and logs the chosen cipher suite, which is then exported to Censys. Using Censys, we generated the distribution of selected cipher suites by all HTTPS hosts by querying for HTTPS hosts and aggregating on the 443.https.cipher_suite.name

University	Protocol	Shodan	Censys
Michigan (141.212.0.0/16)	FTP	38	255
	HTTP	274	987
	HTTPS	53	337
Iowa (128.255.0.0/16)	FTP	12	98
	HTTP	415	1,304
	HTTPS	30	662
Berkeley (128.32.0.0/16)	FTP	84	582
	HTTP	602	1,871
	HTTPS	158	1,188

Table 3.8: **Shodan Comparison** — We compared the number of hosts returned by Shodan and Censys for FTP, HTTP, and HTTPS on three different /16 network blocks, each belonging to a different public U.S. university. We find that, on average, Censys found 600% more FTP hosts, 220% more HTTP hosts, and 800% more HTTPS hosts.

field, which Censys completed in 212ms. We show these distributions in Figure 3.5. Even from these basic distributions, we can gain insights into cipher suite selection in the wild. The Alexa Top 1M domains prefer ECDHE key exchange, whereas IPv4 prefers RSA key exchange. Hosts without trusted certificates are much more likely to use RC4 rather than AES. We can also see that overall not only are RC4 and AES preferred to 3DES, but that 3DES ciphers are only chosen by less than 1% of hosts.

3.7 Related Work

There have been a large number of research studies over the past several years that have been based on Internet-wide scanning [24, 51, 58, 59, 63, 68, 78, 83, 99, 100, 122, 133, 153, 155, 165, 169, 253], which encouraged us to develop Censys.

Censys further enables these types of studies, and lowers the barriers to entry for utilizing Internet-wide scan data. Similarly, there have been several scanners designed for scanning the IPv4 address space, notably ZMap [101] and Masscan [117]. While we introduce ZGrab, an application scanner, Censys itself is not a new Internet scanner. Rather, it builds upon ZMap to provide a higher level interface to scan data.

3.7.1 Scan Driven Search Engines

The closest work to Censys is Shodan, which provides a text search of banners, primarily on FTP, SSH, Telnet, and HTTP [173]. Banners are searchable as plaintext, and searches can be filtered by CIDR range and location. While Censys and Shodan seek to to fulfill similar goals, they take different approaches, offer differing functionality, and fulfill different needs.

Unfortunately, it is unclear how Shodan performs banner grabs, from where, and how frequently. In order to compare Shodan's coverage with Censys, we compared the result sets for three academic institutions on three shared protocols: FTP, HTTP, and HTTPS. Shodan does not expire results, which makes it difficult to compare with our most recent scans. However, Shodan does include a last-seen-at timestamp on records and we compare Censys against any hosts Shodan has seen in the past month. In comparison, Censys only presents records from the latest scan. This will inflate the number of results presented by Shodan, given that hosts frequently move, but allows us to approximate the differences between the two services. On average, Censys found 598% more FTP hosts, 222% more HTTP hosts, and 808% more HTTPS hosts than Shodan. Spot-checks confirmed that these additional hosts were not false positives, but rather hosts not recently found, or were missing from Shodan.

In order to measure how frequently Shodan scans the Internet, we provisioned a publicly available FTP server with a unique banner, and queried the Shodan search API every 10 minutes for the banner. Despite this, Shodan did not respond with our FTP host in its result set until 25 days after provisioning our server. In comparison, the host was present in Censys's public interface in under 48 hours. We also note that during this experiment, Shodan timed out for 2% of queries and returned an Invalid Query error for 1%.

Shodan's search interface and API further differ from Censys. In comparison to Censys's support for query statements on parsed out fields, Shodan only allows simple full-text searches against the raw text of a host's banner. Further, Shodan limits anonymous users to 10 hosts per search and registered users to 50 hosts per search. All API calls require creating an account, and access to larger result sets and HTTPS requires purchasing an account (a minimum 50 USD). Any results from the API and results in the web interface beyond the 50 hosts per search require purchasing "query credits". Credits can be purchased at $2.50/credit, or $500/month for unlimited credits, and allow viewing. In comparison, Censys publicly provides a fully featured query interface to parsed application handshake data and it provides API results in paginated sets of 5k hosts. We further post all raw and parsed data from Censys on the Internet-Wide Scan Data Repository at https://scans.io.

While Shodan has frequently been used to show the existence of vulnerable systems, its lack of timeliness, coverage, and transparency prevents its use as a trusted tool by researchers. In contrast, Censys is architected to be a fully transparent, community-driven project. All of the code is available on GitHub, all results are publicly available, and we support a query syntax and API tailored to researchers.

3.8 Conclusion

Until now, there remained a gap between the technical ability to perform host discovery scans on the IPv4 address space and answering meaningful research questions. In this chapter, we introduced Censys, a public query engine and data processing facility backed by data collected from ongoing Internet-wide scans. Designed to help researchers answer security related questions, Censys collects structured data about the IPv4 address space and supports querying fields derived from scans and generating statistical reports. We explored several security applications of Censys and showed how Censys can be used to easily answer questions from recent studies. We hope that Censys enables researchers to easily answer questions about the Internet that previously required extensive effort, while simultaneously reducing duplicate effort and total scan traffic.

CHAPTER 4

Detecting Widespread Weak Keys in Network Devices

4.1 Introduction and Roadmap

Randomness is essential for modern cryptography, where security often depends on keys being chosen uniformly at random. Researchers have long studied random number generation, from both practical and theoretical perspectives (e.g., [56, 69, 85, 90, 123, 125]), and a handful of major vulnerabilities (e.g., [47, 114]) have attracted considerable scrutiny to some of the most critical implementations. Given the importance of this problem and the effort and attention spent improving the state of the art, one might expect that today's widely used operating systems and server software generate random numbers securely. In this chapter, we test that proposition empirically by examining the public keys in use on the Internet.

The first component of our study is the most comprehensive Internet-wide survey to date of two of the most important cryptographic protocols, TLS and SSH (Section 4.3.1). By scanning the public IPv4 address space, we collected 5.8 million unique TLS certificates from 12.8 million hosts and 6.2 million unique SSH host keys from 10.2 million hosts. This is 67% more TLS hosts than the latest released EFF SSL Observatory dataset [102]. Our techniques take less than 24 hours to scan the entire address space for listening hosts and less than 96 hours to retrieve keys from them. The results give us a macroscopic perspective of the universe of keys.

Next, we analyze this dataset to find evidence of several kinds of problems related to inadequate randomness. To our surprise, at least 5.57% of TLS hosts and 9.60% of SSH hosts use the same keys as other hosts in an apparently vulnerable manner (Section 4.4.1). In the case of TLS, at least 5.23% of hosts use manufacturer default keys that were never changed by the owner, and another 0.34% appear to have generated the same keys as one or more other hosts due to malfunctioning random number generators. Only a handful of the vulnerable TLS certificates are signed by browser-trusted certificate authorities.

Even more alarmingly, we are able to compute the private keys for 64,000 (0.5%) of the TLS hosts and 108,000 (1.06%) of the SSH hosts from our scan data alone by exploiting known weaknesses of RSA and DSA when used with insufficient randomness. In the case of RSA, distinct moduli that share exactly one prime factor will result in public keys that appear distinct but whose private keys are efficiently computable by calculating the greatest common divisor (GCD). We implemented an algorithm that can compute the GCDs of all pairs of 11 million distinct public RSA moduli in less than 2 hours (Section 4.3.3). Using the resulting factors, we are able to obtain the private keys for 0.50% of TLS hosts and 0.03% of SSH hosts (Section 4.4.2). In the case of DSA, if a DSA key is used to sign two different messages with the same ephemeral key, an attacker can efficiently compute the signer's long-term private key. We find that our SSH scan data contain numerous DSA signatures that used the same ephemeral keys during signing, allowing us to compute the private keys for 1.6% of SSH DSA hosts (Section 4.4.3).

To understand why these problems are occurring, we manually investigated hundreds of the vulnerable hosts, which were representative of the most commonly repeated keys as well as each of the private keys we obtained (Section 4.3.2). Nearly all served information identifying them as headless or embedded systems, including routers, server management cards, firewalls, and other network devices. Such devices typically generate keys automatically on first boot, and may have limited entropy sources compared to traditional PCs. Furthermore, when we examined clusters of hosts that shared a key or factor, in nearly all cases these appeared to be linked by a manufacturer or device model. These observations lead us to conclude that the problems are caused by specific defective implementations that generate keys without having collected sufficient entropy. We identified vulnerable devices and software from 55 manufacturers, including some of the largest names in the technology industry, and worked to notify the responsible parties.

In the final component of our study, we experimentally explore the root causes of these vulnerabilities by investigating several of the most common open-source software components from the population of vulnerable devices (Section 4.5). Based on the devices we identified, it is clear that no one implementation is solely responsible, but we are able to reproduce the vulnerabilities in plausible software configurations. Every software package we examined relies on `/dev/urandom` to generate cryptographic keys; however, we find that Linux's random number generator (RNG) can exhibit a boot-time entropy hole that causes `urandom` to produce deterministic output under conditions likely to occur in headless and embedded devices. In experiments with OpenSSL and Dropbear SSH, we show how repeated output from the system RNG can lead not only to repeated long-term keys but also to factorable RSA keys and repeated DSA ephemeral keys due to the behavior of

application-specific entropy pools.

Given the diversity of the devices and software implementations involved, mitigating these problems will require action by many different parties. We draw lessons and recommendations for developers of operating systems, cryptographic libraries, and applications, and for device manufacturers, certificate authorities, end users, and the security and cryptography communities (Section 4.7).

It is natural to wonder whether these results should call into question the security of every RSA or DSA key. Based on our analysis, the margin of safety is slimmer than we might like, but we have no reason to doubt the security of most keys generated interactively by users on traditional PCs. While we took advantage of the details of specific cryptographic algorithms in this chapter, we conclude that the blame for these vulnerabilities lies chiefly with the implementations. Ultimately, the results of our study should serve as a wake-up call that secure random number generation continues to be an unsolved problem in important areas of practice.

Online resources For the most recent version of this paper, partial source code, and our online key-check service, visit https://factorable.net.

4.2 Background

In this section, we review the RSA and DSA public-key cryptosystems and discuss the known weaknesses of each that we used to compromise private keys. We then discuss how an adversary might exploit compromised keys to attack SSH and TLS in practice.

4.2.1 RSA review

An RSA [218] public key consists of two integers: an exponent e and a modulus N. The exponent can be shared among multiple public keys without compromising security, but the modulus cannot. The modulus N is the product of two randomly chosen prime numbers p and q. The private key is the decryption exponent

$$d = e^{-1} \bmod (p-1)(q-1).$$

Anyone who knows the factorization of N can efficiently compute the private key for any public key (e,N) using the preceding equation. When p and q are unknown, the most efficient known method to calculate the private key is to factor N into p and q and use the above equation to calculate d [57].

Factorable RSA keys No one has been publicly known to factor a well-generated 1024-bit RSA modulus; the largest known factored modulus is 768 bits, which was announced in

December 2009 after a multiyear distributed-computing effort [150]. In contrast, the greatest common divisor (GCD) of two 1024-bit integers can be computed in microseconds. This asymmetry leads to a well-known vulnerability: if an attacker can find two distinct RSA moduli N_1 and N_2 that share a prime factor p but have different second prime factors q_1 and q_2, then the attacker can easily factor both moduli by computing their GCD, p, and dividing to find q_1 and q_2. The attacker can then compute both private keys as explained above.

4.2.2 DSA review

A DSA [166] public key consists of three so-called domain parameters (two prime moduli p and q and a generator g of the subgroup of order q mod p) and an integer $y = g^x \bmod p$, where x is the private key. The domain parameters may be shared among multiple public keys without compromising security. A DSA signature consists of a pair of integers (r,s): $r = g^k \bmod p \bmod q$ and $s = (k^{-1}(H(m) + xr)) \bmod q$, where k is a randomly chosen ephemeral private key and $H(m)$ is the hash of the message.

Low-entropy DSA signatures DSA is known to fail catastrophically if the ephemeral key k used in the signing operation is generated with insufficient entropy [46]. (Elliptic curve DSA (ECDSA) is similarly vulnerable. [62]) If k is known for a signature (r,s), then the private key x can be computed from the signature and public key as follows:

$$x = r^{-1}(ks - H(m)) \bmod q.$$

If a DSA private key is used to sign two different messages with the same k, then an attacker can efficiently compute the value k from the public key and signatures and use the above equation to compute the private key x [160]. If two messages m_1 and m_2 were signed using the same ephemeral key k to obtain signatures (r_1,s_1) and (r_2,s_2), then this will be immediately clear as r_1 and r_2 will be equal. The ephemeral key k can be computed as:

$$k = (H(m_1) - H(m_2))(s_1 - s_2)^{-1} \bmod q.$$

4.2.3 Attack scenarios

The weak key vulnerabilities we describe in this paper can be exploited to compromise two of the most important cryptographic transport protocols used on the Internet, TLS and SSH, both of which commonly use RSA or DSA to authenticate servers to clients.

TLS In TLS [87], the server sends its public key in a TLS certificate during the protocol handshake. The key is used either to provide a signature on the handshake (when Diffie-Hellman key exchange is negotiated) or to encrypt session key material chosen by the client

(when RSA-encrypted key exchange is negotiated).

If the key exchange is RSA encrypted, a passive eavesdropper with the server's private key can decrypt the message containing the session key material and use it to decrypt the entire session. If the session key is negotiated using Diffie-Hellman key exchange, then a passive attacker will be unable to compromise the session key from just a connection transcript. However, in both cases, an active attacker who can intercept and modify traffic between the client and server can man-in-the-middle the connection in order to decrypt or modify the traffic.

SSH In SSH, host keys allow a server to authenticate itself to a client by providing a signature during the protocol handshake. There are two major versions of the protocol. In SSH-1 [256], the client encrypts session key material using the server's public key. SSH-2 [257] uses a Diffie-Hellman key exchange to establish a session key. The user manually verifies the host key fingerprint the first time she connects to an SSH server. Most clients then store the key locally in a `known_hosts` file and automatically trust it for all subsequent connections.

As in TLS, a passive eavesdropper with a server's private key can decrypt an entire SSH-1 session. However, because SSH-2 uses Diffie-Hellman, it is vulnerable only to an active man-in-the-middle attack. In the SSH user authentication protocol, the user-supplied password is sent in plaintext over the encrypted channel. An attacker who knows a server's private key can use the above attacks to learn a user's password and escalate an attack to the system.

4.3 Methodology

In this section, we explain how we performed our Internet-wide survey of public keys, how we attributed vulnerable keys to devices, and how we efficiently factored poorly generated RSA keys.

4.3.1 Internet-wide scanning

We performed our data collection in three phases: discovering IP addresses accepting connections on TCP port 443 (HTTPS) or 22 (SSH); performing a TLS or SSH handshake and storing the presented certificate chain or host key; and parsing the collected certificates and host keys into a relational database. Table 4.1 summarizes the results.

Host discovery In the first phase, we scanned the public IPv4 address space to find hosts with port 443 or 22 open. We used the Nmap 5 network exploration tool to perform a SYN scan[1], which involves sending a TCP SYN packet to each candidate host and detecting

[1]The Nmap options we used were: `-sS -Pn -n -T5 --min-hostgroup=2000`

	SSL Observatory (12/2010)	Our TLS scan (10/2011)	Our SSH scans (2-4/2012)
Hosts with open port 443 or 22	≈16,200,000	28,923,800	23,237,081
Completed protocol handshakes	7,704,837	12,828,613	10,216,363
Distinct RSA public keys	3,933,366	5,656,519	3,821,639
Distinct DSA public keys	1,906	6,241	2,789,662
Distinct TLS certificates	4,021,766	5,847,957	—
Trusted by major browsers	1,455,391	1,956,267	—

Table 4.1: **Internet-wide scan results** — We exhaustively scanned the public IPv4 address space for TLS and SSH servers listening on ports 443 and 22, respectively. Our results constitute the largest such network survey reported to date. For comparison, we also show statistics for the EFF SSL Observatory's most recent public dataset [102].

whether the host responds with a SYN-ACK packet. We chose this scanning method based on its low bandwidth requirements; for the vast majority of hosts (those with the target port closed), at most two packets need to be exchanged. We excluded address ranges reserved for location identification, private use, loopback, link local, multicast, or future use in the IANA IPv4 address space registry.

We executed our first host discovery scan beginning on October 6, 2011 from 25 Amazon EC2 Micro instances spread across five EC2 regions (Virginia, California, Japan, Singapore, and Ireland). The scan ran at an average of 40,566 IPs/second and finished in 25 hours; it found 28,923,800 IPs with port 443 open. We performed a second scan to find SSH hosts, beginning on February 12, 2012; it found 23,237,081 IPs with port 22 open.

Certificate and host-key retrieval In the second phase of the scanning process, we attempted a TLS or SSH protocol handshake with the addresses accepting connections on port 443 or 22 and recorded the TLS certificate or SSH host key presented by the server.

For TLS, we implemented a certificate fetcher in Python using the Twisted event-driven network framework [148]. We fetched TLS certificates using an EC2 Large instance with five processes each maintaining 800 concurrent connections. We started fetching certificates on October 11, 2011. The fetcher processed a mean of 83 hosts/second and completed in approximately 96 hours. It retrieved certificate chains from 12,828,613 IP addresses (44.4% of hosts listening on port 443). These certificate chains contained 5,656,519 distinct RSA public keys and 6,241 distinct DSA public keys.

To efficiently collect SSH host keys, we implemented a simple SSH client in C, which is able to process upwards of 1200 hosts/second by concurrently performing protocol

`--max-rtt-timeout=500ms --min-rate=10000`[2]. We selected these parameters by choosing the most aggressive parameters that did not cause non-negligible dropoff in response on an EC2 Micro Instance.

handshakes using *libevent* [174]. Our client initiates a normal-looking connection and offers only `Diffie-Hellman Group 1` key exchange[3], and then downloads and stores the host key provided by the server.

Initially, we ran the fetcher from an EC2 Large instance in a run that started on February 12, 2012. This run targeted only RSA-based host keys and returned 3,821,639 distinct keys from 10,216,363 IP addresses (44.0% of hosts listening on port 22). In two later runs, we targeted DSA-based host keys, and rescanned those hosts that had offered DSA keys in the first SSH scan. For these, we also stored the authentication signature provided by the server; we varied the client string to ensure that each signature would be distinct. The first DSA run started on March 26, 2012 from a host at UCSD; it returned 2,091,643 distinct DSA host keys from 4,572,218 IP addresses (51.4% of listening hosts). The second run, from a host at the University of Michigan, started on April 1, 2012; it took 3 hours to complete and returned 2,058,706 distinct DSA host keys from 4,542,707 IP addresses.

TLS certificate processing For TLS, we performed a third processing stage in which we parsed the previously fetched certificate chains and generated a database from the X.509 fields. We implemented a certificate parser in Python and C primarily based on the M2Crypto [226] SWIG [44] interface to the OpenSSL library [79]. In total, we found 5,847,957 distinct certificates, of which 1,956,267(33.5%) were browser trusted. We deemed a certificate to be trusted if we could find a chain of trust leading to a root CA trusted by the current version of Firefox, MacOS, OpenSSL, or Windows.

4.3.2 Identifying vulnerable device models

We attempted to determine what hardware and software generated or served the weak keys we identified using manual detective work. The most straightforward method was based on TLS certificate information—predominately the X.509 *subject* and *issuer* fields. In many cases, the certificate identified a specific manufacturer or device model. Other certificates contained less information; we attempted to identify these devices through Nmap host detection or by inspecting the public contents of HTTPS sites or other IP services hosted on the IP addresses.

When we could identify a pattern in vulnerable TLS certificates that appeared to belong to a device model or product line, we constructed regular expressions to find other similar devices in our scan results. Under the theory that the keys were vulnerable because of a problem with the design of the devices (where they were most likely generated), this allows us to estimate the total population of devices that might be potentially vulnerable, beyond

[3] Our SSH client implementation is greatly simplified by only supporting the `Diffie-Hellman Group 1` key exchange, which is supported by all SSH servers.

those serving immediately compromised keys.

Identifying SSH devices was more problematic, as SSH keys do not include descriptive fields and the server identification string used in the protocol often indicated only a common build of a popular SSH server. We were able to classify many of the vulnerable SSH hosts using a combination of TCP/IP fingerprinting and examination of information served over HTTP and HTTPS.

When we were able to identify a vulnerable device model, we attempted to disclose the problem to the manufacturer. We provided these manufacturers with background on the vulnerabilities, selected vulnerable IP addresses from among the hosts we identified, and suggestions for remedying the problem.

The device names and manufacturers that we report here have been identified with moderate or high confidence given the available information. However, because we do not have physical access to the hosts, we cannot state with certainty that all our identifications are correct.

4.3.3 Efficiently computing all-pairs GCDs

We now describe how we efficiently computed the pairwise GCD of all distinct RSA moduli in our multimillion-key dataset. This allowed us to calculate RSA private keys for 66,540 vulnerable hosts that shared one of their RSA prime factors with another host in our survey.

The fastest known factoring method for general integers is the number field sieve, which has heuristic complexity $O(2^{n^{1/3}(\log n)^{2/3}})$ for n-bit numbers [161]. With current computing power, factoring any of the 85,988 512-bit RSA public keys detected in our scan is within the reach of a dedicated amateur using existing implementations and common hardware [247], but factoring *all of them* would require thousands of years of computation.

In contrast to factoring, the greatest common divisor (GCD) of two integers can be computed very efficiently using Euclid's algorithm, with computational complexity $O(n^2)$ bits for n-bit numbers. Using fast integer arithmetic, the complexity of GCD can be improved to $O(n(\lg n)^2 \lg\lg n)$ [50]. Computing the GCD of two 1024-bit RSA moduli using the GMP library [119] takes approximately 15 μs on a current mid-range computer.

The naïve way to compute the GCDs of every pair of integers in a large set would be to apply a GCD algorithm to each pair individually. There are 6×10^{13} distinct pairs of RSA moduli in our data; at 15 μs per pair, this calculation would take 30 years. We can do much better by using a more efficient algorithm.

To accomplish this, we implemented a quasilinear-time algorithm for factoring a collection of integers into coprimes, due to Bernstein [49]. The relevant steps, illustrated in Figure 4.1, are as follows:

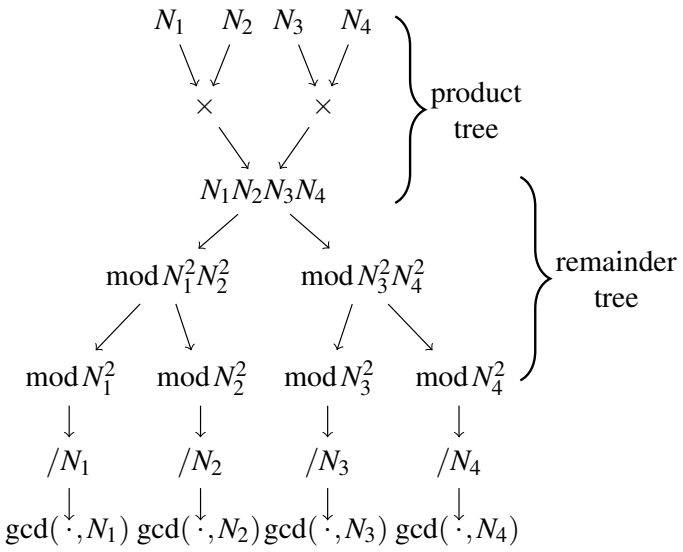


Figure 4.1: **Computing all-pairs GCDs efficiently** — We computed the GCD of every pair of RSA moduli in our dataset using an algorithm due to Bernstein [49]: First, compute the product of all moduli using a product tree, then use a remainder tree to reduce the product modulo each input. The final output is the GCD of each input modulus with the product of all the other moduli.

Algorithm 1 Quasilinear GCD finding

Input: $N_1, \ldots, N_m$ RSA moduli

1: Compute $P = \prod N_i$ using a product tree.

2: Compute $z_i = (P \bmod N_i^2)$ for all i using a remainder tree.

Output: $\gcd(N_i, z_i/N_i)$ for all i.

A product tree computes the product of m numbers by constructing a binary tree of products. A remainder tree computes the remainder of an integer modulo many integers by successively computing remainders for each node in their product tree. For further discussion, see Bernstein [50].

The final output of the algorithm is the GCD of each modulus with the product of all the other moduli. We are interested in the moduli for which this GCD is not 1. However, if a modulus shares both of its prime factors with two other distinct moduli, then the GCD will be the modulus itself rather than one of its prime factors—therefore providing us no immediate factorization. This occurred in a handful of instances in our dataset; we factored these moduli using the naïve quadratic algorithm for pairwise GCDs.

	Our TLS Scan		Our SSH Scans	
Number of live hosts	12,828,613	(100.00%)	10,216,363	(100.00%)
... using repeated keys	7,770,232	(60.50%)	6,642,222	(65.00%)
... using vulnerable repeated keys	714,243	(5.57%)	981,166	(9.60%)
... using default certificates or default keys	670,391	(5.23%)		
... using low-entropy repeated keys	43,852	(0.34%)		
... using RSA keys we could factor	64,081	(0.50%)	2,459	(0.03%)
... using DSA keys we could compromise			105,728	(1.03%)
... using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
... using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
... identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
... model using low-entropy repeated keys	314,640	(2.45%)		

Table 4.2: **Summary of vulnerabilities** — We analyzed our TLS and SSH scan results to measure the population of hosts exhibiting several entropy-related vulnerabilities. These include use of repeated keys, use of RSA keys that were factorable due to repeated primes, and use of DSA keys that were compromised by repeated signature randomness. Under the theory that vulnerable repeated keys were generated by embedded or headless devices with defective designs, we also report the number of hosts that we identified as these device models. Many of these hosts may be at risk even though we did not specifically observe repeats of their keys.

Using fast integer arithmetic algorithms, multiplication and modular reduction on n-bit integers can be done in time $O(n \lg n \lg\lg n)$, and GCDs in time $n(\lg n)^2 \lg\lg n$ [50]. Thus, the asymptotic running time of Algorithm 1 on m n-bit integers would be $O(mn \lg m \lg(mn) \lg\lg(mn))$ for the product and remainder trees and $O(mn(\lg n)^2 \lg\lg n)$ to compute the GCDs.

We implemented the algorithm in C using the GMP library [119] for the arithmetic operations and ran it on the 11,170,883 distinct RSA moduli from our TLS and SSH datasets and the EFF SSL Observatory [102] dataset. Our product tree had 24 levels, each of which was about 2 GB in size. In addition, GMP allocates a large amount of scratch memory during the calculations—around 30 GB for our dataset. Due to memory limitations, we wrote each level to disk. We found that the product of all of the moduli was too large for GMP's raw-integer I/O format, which cannot process numbers larger than 2^{31} bytes. We patched the library to remove this limitation.

The entire computation finished in 5.5 hours using a single core on a machine with a 3.30 GHz Intel Core i5 processor and 32 GB of RAM. The remainder tree took approximately ten times as long to process as the product tree. Parallelized across sixteen cores on an EC2 Cluster Compute Eight Extra Large Instance with 60.5 GB of RAM and using EBS-backed storage for scratch data, the same computation took 1.3 hours at a cost of about $5.

4.4 Vulnerabilities

We analyzed the data from our TLS and SSH scans and identified several patterns of vulnerability that would have been difficult to detect without a macroscopic view of the Internet. This section discusses the details of these problems, as summarized in Table 4.2.

4.4.1 Repeated keys

We found that 7,770,232 of the TLS hosts (61%) and 6,642,222 of the SSH hosts (65%) served the same key as another host in our scans. To understand why, we clustered certificates and host keys that shared the same public key and manually inspected representatives of the largest clusters. In all but a few cases, the TLS certificate subjects, SSH version strings, or WHOIS information were identical within a cluster, or pointed to a single manufacturer or organization. This sometimes suggested an explanation for the shared keys.

Not all of the repeated keys were due to vulnerabilities. For instance, many of the most commonly repeated keys appeared in shared hosting situations. Six of the ten most common DSA host keys and three of the ten most common RSA host keys were served by large hosting providers (see Figure 4.2). Another frequent reason for repeated keys was distinct TLS certificates all belonging to the same organization. For example, TLS hosts at google.com, appspot.com, and doubleclick.net all served distinct certificates with the same

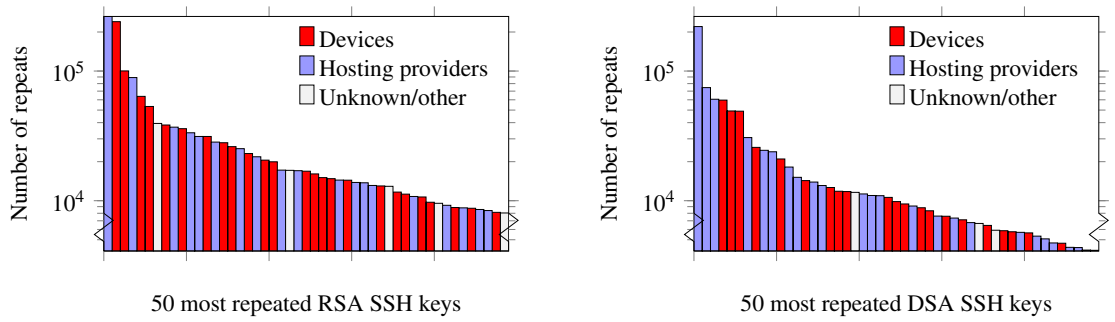


Figure 4.2: **Commonly repeated SSH keys** — We investigated the 50 most repeated SSH host keys for both RSA and DSA. Nearly all of the repeats appeared to be due either to hosting providers using a single key on many IP addresses or to devices that used a default key or generated keys using insufficient entropy. Note log scale.

public key. We excluded these cases and attributed remaining clusters of shared keys to several classes of problems.

Default keys A common reason for hosts to share the same key that we *do* consider a vulnerability is manufacturer-default keys. These are preconfigured in the firmware of many devices, such that every device of a given model shares the same key pair unless the user changes it. The private keys to these devices may be accessible through reverse engineering, and published databases of default keys such as littleblackbox [129] contain private keys for thousands of firmware releases.

At least 670,391 (5.23%) of the TLS hosts appeared to serve manufacturer-default certificates or keys. We identified 57 distinct TLS certificates from the littleblackbox 0.1.3 database, revealing private keys for 23,476 (0.18%) of the TLS hosts. We were able to identify 170 additional default certificates by manually inspecting the 654 self-signed certificates that appeared on more than 250 IP addresses. (Most browser-trusted certificates that were served on multiple IP addresses appeared to be legitimate HTTPS sites served from multiple web servers.) We classified a certificate as a manufacturer default if nearly all the devices of a given model used identical certificates, or if the certificate was labeled as a default certificate. Our manually identified manufacturer-default certificates were served by another 618,014 (4.82%) of the TLS hosts.

The most common default certificate that we could ascribe to a particular device belonged to a model of consumer router. Our scan uncovered 90,779 instances of this device model sharing a single certificate. We also found a variety of enterprise products serving default keys, including server management devices, network storage devices, routers, remote access devices, and VoIP devices.

At least another 85,046 TLS hosts (0.66%) served default Apache certificates (sometimes referred to as snake-oil certificates, because they often include the CN `www.snakeoil.dom`);

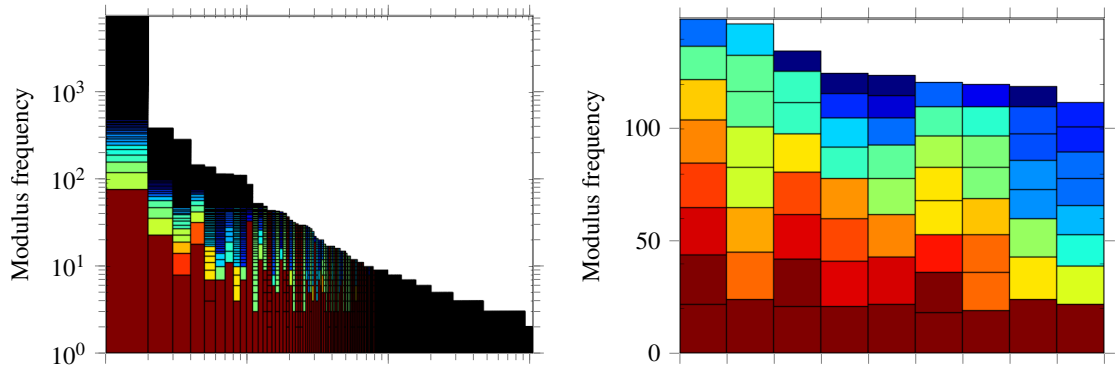
we identified 832 distinct snake-oil certificates that were served on more than one host, as many Linux distributions ship distinct default certificates. We did not count these among the vulnerable manufacturer-default keys. However, we also found 58 non-default certificates that shared an RSA modulus with one of these snake-oil certificates, including 22 that had been signed by browser-trusted CAs. In these cases, it is possible that organizations generated certificates with their own information but included the default public key shipped with Apache into their own certificate, rather than generating a new key pair.

We observed a similar phenomenon with default TLS keys served by Citrix remote access servers: we found 1584 apparently unrelated Citrix hosts serving distinct certificates (including 16 signed by trusted CAs) that shared keys with default certificates. The signed certificates belonged to entities including Fortune 500 companies, insurance providers, law firms, a major public transit authority, and the U.S. Navy. These organizations may be authenticating or encrypting remote-access service communication using these duplicate keys.

For most of the repeated SSH keys, the lack of uniquely identifying host information prevents us from distinguishing default keys from keys generated with insufficient entropy, so we address these together in the next section.

Repeated keys due to low entropy Another common reason that hosts share the same key appears to be entropy problems during key generation. In these instances, when we investigated a key cluster, we would typically see thousands of hosts across many address ranges, and, when we checked the keys corresponding to other instances of the same model of device, we would see a long-tailed distribution in their frequencies. Intuitively, this type of distribution suggests that the key generation process may be using insufficient entropy, with distinct keys due to relatively rare events. For TLS, our investigations began with the keys that occurred in at least 100 distinct self-signed certificates. For SSH, we started from the 50 most commonly repeated keys for each of RSA (appearing on more than 8000 hosts) and DSA (appearing on more than 4000 hosts).

With this process, we identified 43,852 TLS hosts (0.34%) that served repeated keys due apparently to low entropy during key generation. The repeated keys included 208 distinct TLS RSA keys. 27,545 certificates (98%) containing these repeated keys were self-signed; all 577 CA-signed certificates identified Iomega StorCenter network storage devices. For most SSH hosts we were unable to distinguish between default keys and keys repeated due to entropy problems, but 981,166 of the SSH hosts (9.60%) served keys repeated for one of these reasons. Our TLS and SSH datasets contained two RSA keys in common; one was served from the same host on both TLS and SSH, and the other was served by TLS and SSH on distinct and seemingly unrelated hosts.



(a) Primes generated by Juniper network security devices

(b) Primes generated by IBM remote management cards

Figure 4.3: **Visualizing RSA common factors** — Different implementations displayed different patterns of vulnerable keys. These plots depict the distribution of vulnerable keys divisible by common factors generated by two different device models. Each column represents a collection of RSA moduli divisible by a single prime factor p. The color and internal rectangles show, for each p, the frequencies of each distinct prime factor q. The Juniper device (*left*; note log-log scale) follows a long-tailed distribution that is typical of many of the devices we identified. In contrast, the IBM remote access device (*right*) was unique among those we observed in that it generates RSA moduli roughly uniformly distributed among nine distinct prime factors.

We used the techniques described in Section 4.3.2 to identify apparently vulnerable devices from 27 manufacturers. These include enterprise-grade routers from Cisco; server management cards from Dell, Hewlett-Packard, and IBM; VPN devices; building security systems; network attached storage devices; and several kinds of consumer routers and VoIP products.

Given cryptographic key sizes, we would not expect to see devices generate a single duplicated key for the population sizes we examined if the keys were generated with sufficient entropy. Therefore, duplicated keys are a red flag that calls the security of the device's key generation process into question, and all keys generated with the same model device should be considered suspect. For 14 of the TLS devices generating repeated keys, we were able to infer a fingerprint for the device model and estimate the total population of the device in our scan. The prevalence of duplicated keys varied greatly for different device models, from as low as 0.2% in the case of one router to 98% for one thin client. The total population of these identified, potentially vulnerable TLS devices was 314,640 hosts, which represents 2.45% of the TLS hosts in our scan.

In the above analyses, we excluded repeated keys that were due to the infamous Debian weak-key vulnerability [47, 254]. 4,147 (0.03%) of the hosts in our TLS dataset served

certificates with Debian weak keys. 17 of the 2,904 vulnerable certificates were signed by browser-trusted authorities and 11 were signed after the vulnerability was announced in August 2008. 31,111 (0.34%) of the RSA SSH hosts and 22,030 (0.34%) of the DSA SSH hosts served Debian weak keys. The most commonly repeated weak key appeared on 3,422 SSH RSA hosts and 2,357 SSH DSA hosts. The most repeated Debian weak key in a TLS certificate occurred in 1,115 distinct certificates; the next most common keys appeared in 4 and 2 certificates.

4.4.2 Factorable RSA keys

A second way that entropy problems might manifest themselves during key generation is if an RSA modulus shares one of its prime factors p or q with another key. As explained in Section 4.2.1, finding such a pair immediately allows an adversary to efficiently factor both moduli and obtain their respective private keys. In order to find such keys, we computed the GCD of all pairs of distinct RSA moduli by applying the algorithm described in Section 4.3.3 to a combined dataset consisting of the keys in our TLS and SSH scans and the EFF SSL Observatory data. Running the algorithm on the combined datasets allowed us to factor more keys than performing the computation separately for each, since with a larger collection of inputs the algorithm has a larger factor base to use on each key.

The 11,170,883 distinct RSA moduli yielded 2,314 distinct prime divisors, which divided 16,717 distinct public keys. This allowed us to obtain private keys for 23,576 (0.40%) of the TLS certificates in our scan data, which were served on 64,081 (0.50%) of the TLS hosts, and 1,013 (0.02%) of the RSA SSH host keys, which were served on 2,459 (0.027%) of the RSA SSH hosts.

The vast majority of the vulnerable keys appeared to be system-generated certificates and SSH host keys used by routers, firewalls, remote administration cards, and other types of headless or embedded network devices. Only two of the factorable TLS certificates had been signed by a browser trusted authority and both have expired. Some devices generated factorable keys both for TLS and SSH, and a handful of devices shared common factors across SSH and TLS keys.

We classified these factorable keys in a similar manner to the repeated keys. We clustered the factorable certificates and host keys together by prime divisor. We found that, in all but a small number of cases, the TLS certificates and SSH host keys divisible by a common factor all belonged to a particular manufacturer, which we were able to identify in most cases using the techniques described in Section 4.3.2. We then grouped these clusters by manufacturer and compared them to other certificates and host keys from the same device model.

We identified devices from 41 manufacturers in this way, which constituted 99% of the

hosts that generated RSA keys we could factor. The devices range from 100% (576 of 576 devices) vulnerable to 0.01% vulnerable (2 out of 10,932). As with repeated keys, we would not expect to see well-generated cofactorable keys; any device model observed generating factorable keys should be treated as potentially vulnerable.

The majority of the devices serving factorable keys were Juniper Networks Branch SRX devices[4]. We identified 46,993 of these devices in our dataset, and we factored the keys for 12,688 (27%) of them. Of these keys, 7,510 (59%) share a single common divisor. The distribution of factors among its moduli is shown in Figure 4.3a.

The most remarkable devices were IBM Remote Server Administration cards and BladeCenter devices, which displayed a distribution of factors unlike any of the other vulnerable devices we found. There were only 9 distinct prime factors that had been used to generate the keys for 576 devices. Each device's key was the product of two different primes from this list. The 36 possible moduli that could be generated with this process were roughly uniformly distributed, as shown in Figure 4.3b. In addition, this was the only device we observed to generate RSA moduli where *both* prime factors were shared with other keys.

Several of the prime factors of the vulnerable keys displayed evidence of further entropy problems during key generation. Some shared an improbably large number of most significant bits in common—86 Sentry devices had common divisors sharing 160 out of 512 bits and two 2wire devices had a pair of common divisors with their 85 most significant bits in common. Alarmingly, we observed several prime factors of SSH host keys that were almost all zero bytes, with only the first two and last three bytes set, but we could not identify the device models. In addition to signaling that the primes themselves were generated with insufficient entropy, these constitute a separate, serious vulnerability, as it is possible to efficiently factor an RSA modulus if a large enough fraction of the bits of one of its factors is known [77] or shared with another modulus [176].

Approximately 12 of the SSH RSA keys were divisible by many small prime factors. Based on unlikely bit patterns in the keys, we hypothesize that these cases were due to memory corruption on the devices rather than broken primality testing.[5]

4.4.3 DSA signature weaknesses

The third class of entropy-related vulnerability that we searched for was repeated ephemeral keys in DSA signatures. As explained in Section 4.2.2, if a DSA key is used to sign two different messages using the same ephemeral key, then the long-term private key is immedi-

[4] Juniper identified these models in private communication.

[5] The EFF SSL Observatory data contains several keys divisible by many small prime factors, but we determined that those were due to data corruption during scanning, as none of the cases was repeatable when the hosts were rescanned.

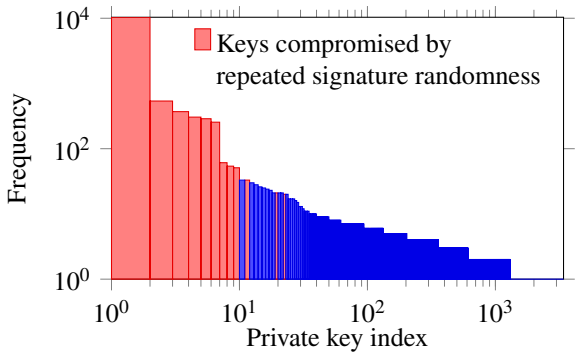


Figure 4.4: **Vulnerable DSA keys for one SSH device** — We identified 18,684 SSH DSA keys that appeared to have been generated by Gigaset DSL routers, of which 16,575 were repeated at least once. Shown in red in this log-log plot are 12,378 keys further compromised due to repeated DSA signature values. The more commonly repeated DSA keys were more likely to be compromised by repeated signatures, as they are more likely to have collisions between hosts in subsequent scans.

ately computable from the public key and the signatures. The presence of this vulnerability indicates entropy problems at later phases of operation, after initial key generation. We searched for signatures from identical keys containing repeated r values. Then we used the method in Section 4.2.2 to compute the corresponding private keys.

Our combined SSH DSA scans collected 9,114,925 signatures (in most cases, two from each SSH host serving a DSA-based key) of which 4,365 (0.05%) contained the same r value as at least one other signature. 4,094 of these signatures (94%) used the same r value and the same public key. This allowed us to compute the 281 distinct private keys used to generate these signatures. These compromised keys were served by 105,728 (1.6%) of the SSH DSA hosts in our combined scans.

We clustered the vulnerable signatures by r values and manufacturer. 2,026 (46%) of the colliding r values appeared to have been generated by Gigaset SX762 consumer-grade DSL routers and revealed private keys for 17,349 (66%) of the 26,374 hosts we identified as this device model (see Figure 4.4). Another 934 signature collisions appeared to be from ADTran Total Access business-grade phone/network routers and revealed private keys for 62,807 (73%) out of 86,301 such hosts. Several vulnerable device models, including the IBM RSA II remote administration cards, also generated factorable RSA keys.

In addition to device randomness issues, we also discovered compromised keys due to flawed software implementations of DSA running on general-purpose servers. For example, hosts using the Java SSH Maverick library always used the same r value for every DSA signature.

We did not collect DSA handshake signatures during our TLS scan, but we did check

the 6,107 DSA signatures that were applied to TLS certificates by signature authorities or self-signing. We found no repeated DSA public keys or signature ephemeral keys in this limited sample.

While we collected multiple signatures from some SSH hosts, 3,917 (89.7%) out of 4,365 of the collisions were from *different* hosts that had generated the same long-term key and also used the same ephemeral key during the key exchange protocol. This problem compounds the danger of the repeated key vulnerability: a single signature collision between any pair of hosts sharing the same key at any point during runtime reveals the private key for every host using that key. In our dataset we observed tens of thousands of hosts using the same public key. While a single host may never repeat an ephemeral key, two hosts sharing a private key can put each others' keys at risk.

We note that any estimation of vulnerability based on our data is an extreme lower bound, as we gathered at most two signatures from each host in our scans. It is likely that many more private keys would be revealed if we collected additional signatures.

4.5 Experimental Investigation

Based on the results the previous section, we conjectured that the problems we observed were an implementational phenomenon. To more deeply understand the causes, we augmented our data analysis with experimental investigation of specific implementations. While there are many independently vulnerable implementations, we chose to examine three open-source cryptographic software components that appeared frequently in the vulnerable populations.

4.5.1 Weak entropy and the Linux RNG

We conjectured that the cause for many of the entropy problems we observed began with insufficient randomness provided by the operating system. This led us to take an in-depth look at the Linux random number generator (RNG). We note that not every vulnerable key was generated on Linux; we also observed vulnerable keys on FreeBSD and Windows systems, and similar vulnerabilities to those we describe here have been reported with BSD's `arc4random` [251].

The Linux RNG maintains three entropy pools, each with an associated counter that estimates how much fresh entropy it has available. The RNG provides a function to mix data into the pool (using a twisted generalized feedback shift register) and to extract entropy from a pool (by hashing the pool contents, mixing the hash back into the pool, then hashing the hash again). Fresh entropy from unpredictable kernel sources is periodically mixed into the *Input pool*. When processes read from `/dev/random`, the kernel extracts the requested amount of entropy from the Input pool and mixes it into the *Blocking pool*, then extracts

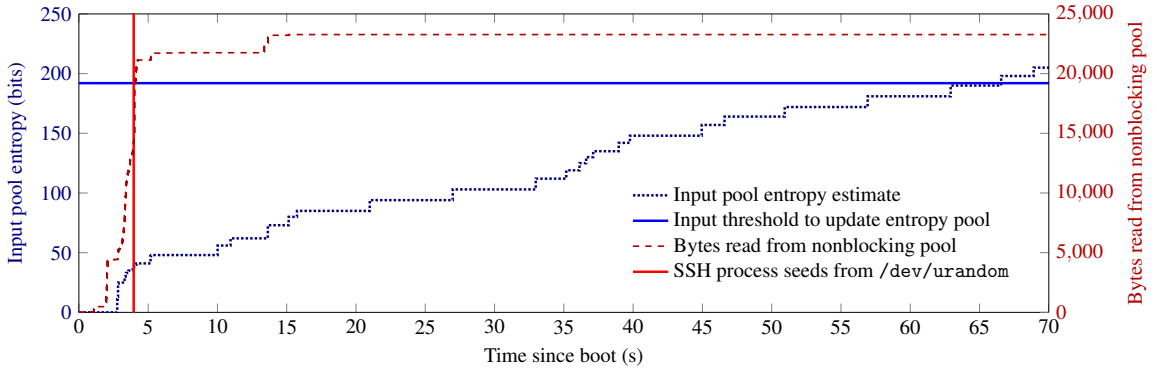


Figure 4.5: **Linux urandom boot-time entropy hole** — We instrumented an Ubuntu Server 10.04 system to record its estimate of the entropy contained in the Input entropy pool during a typical boot. Linux does not mix Input pool entropy into the Nonblocking pool that supplies `/dev/urandom` until the Input pool exceeds a threshold of 192 bits (blue horizontal line), which occurs here at 66 seconds post-boot. For comparison, we show the cumulative number of bytes generated from the Nonblocking entropy pool; the vertical red line marks the time when OpenSSH seeds its internal PRNG by reading from `urandom`, *well before* this facility is ready for secure use.

bytes from the Blocking pool to return to the process. If the Input pool does not contain enough entropy to satisfy the request, the read blocks until additional entropy becomes available. Similarly, when processes read from `/dev/urandom`, the kernel attempts to transfer the requested amount of entropy from the Input pool to the *Nonblocking pool*. In this case, the read will be satisfied immediately using the contents of the Nonblocking pool, even if the Input pool entropy is depleted.

Entropy sources We experimented with the Linux 2.6.35 kernel to exhaustively determine the sources of entropy used by the RNG. To do this, we traced through the kernel source code and systematically disabled entropy sources until the RNG output was repeatable. All of the entropy sources we found are greatly weakened under certain operating conditions.

When the kernel starts, the buffers that contain the pools are deliberately left uninitialized and may contain data left over from the previous boot; however, their contents will be predictable if the system has been powered off long enough for memory to return to its ground state [127] or in systems with ECC RAM, which is zeroized on startup. The startup clock time, in nanosecond resolution, is also mixed in, but this value provides little entropy in systems that do not have a working real-time clock or on the first boot in systems where the real-time clock value is set at zero during manufacturing. It may also be predictable based on system uptime, which is visible to remote attackers via TCP timestamps.

Entropy from the timing of human input events and disk access is also mixed into the Input pool whenever these events occur. However, human input entropy is not available

on unattended systems. Disk timings are only available relatively late in the kernel boot process, after disks have been initialized, and they are not available in embedded devices that lack traditional disk storage.

In addition, typical Linux distributions save entropy across boots by copying bytes from `urandom` to a local file on shutdown and reading them into the Blocking and Nonblocking pools at startup. However, this file is not present on the first system boot, and it will not be available until after the filesystem has been mounted.

The final and most interesting entropy source was one that we have not seen documented elsewhere. Whenever the kernel extracts entropy from a pool, it hashes the pool contents and then mixes part of the result back into the pool. The developers chose not to put a lock around this entire procedure, and, as a result, if two threads extract entropy concurrently, the pool contents may change anywhere in the middle of the hash computation. With eight physical threads, we observed on average approximately 100 instances of re-entrant entropy extraction from the Nonblocking pool during startup, mainly from entropy consumption within the kernel. There was a high degree of variation to the exact timing of these calls, resulting in the introduction of significant (but uncredited) entropy to the pool. Reentrant calls dropped to almost zero when we forced the kernel to use only one physical thread (with the `maxcpus=0` flag), and we did not observe any entropy being induced.

Surprisingly, modern Linux systems no longer collect entropy from IRQ timings. The Linux kernel maintainers deprecated the use of this source in 2009 by removing the `IRQF_SAMPLE_RANDOM` flag, apparently to prevent events controlled or observable by an attacker from contributing to the entropy pools. Although mailing list discussions suggest the intent to replace it with new collection mechanisms tailored to each interrupt-driven source [112], as of Linux 3.4.2 no such functions have been added to the kernel. The removal of IRQs as an entropy source has likely exacerbated RNG problems in headless and embedded devices, which often lack human input devices, disks, and multiple cores. If they do, the only source of entropy—if there are any at all—may be the time of boot.

Experiment To test whether Linux's `/dev/urandom` can produce repeatable output in conditions resembling the initial boot of a headless or embedded networked device, we modified the 2.6.35 kernel to add instrumentation to the RNG and disable certain entropy sources: (1) we zeroized the entropy pools during initialization to simulate a cold boot with constant memory state; (2) we used a fixed time value instead of the actual realtime clock value to simulate a machine without a working clock; (3) we limited the kernel to one physical thread to simulate a low-end CPU.

We experimented with this kernel on a Dell Optiplex 980 system using a fresh installation of Ubuntu server 10.04.4. The machine was configured with a Core i7 CPU, 4 GB RAM, a

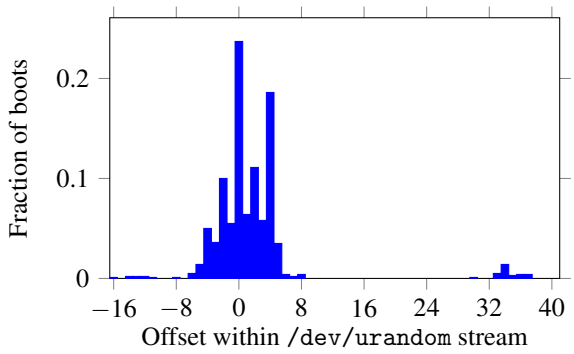


Figure 4.6: **Predictable output from urandom** — When we disabled entropy sources that might be unavailable on a headless or embedded device, the Linux RNG produced the same predictable stream on every boot. The only variation we observed over 1,000 boots was the *position* in the stream where `sshd` read from `urandom`.

32 GB SSD, and a USB keyboard. It was attached to a university office LAN and obtained an IP address using DHCP. With this configuration, we performed 1,000 unattended boots. Each time, we read 32 bytes from `urandom` at the point in the initialization process where the SSH server would normally start. Under these conditions, we found that the output of `/dev/urandom` was entirely predictable and repeatable.

The kernel maintains a reserve threshold for the Input pool, and no data is copied into the Nonblocking pool until the Input pool has been credited with at least that much entropy (192 bits, for our kernel). Figure 4.5 shows the cumulative amount of entropy credited to the Input pool during a typical bootup from our tests. (Note that none of the entropy sources we disabled would have resulted in more entropy being *credited* to the pool.) The credited entropy does not cross this reserve threshold until more than a minute after boot, well after the SSH server and other startup processes have launched. Although Ubuntu tries to restore entropy saved during the last shutdown, this happens slightly *after* the point when `sshd` first reads from `urandom`.

With no entropic inputs, `urandom` produces a deterministic output stream.We did observe variation due to nondeterministic behavior at boot that influences exactly how much data has been read from the stream at the point that the SSH server starts; for instance, other startup processes launched concurrently with the server may also be vying to read from the device. Figure 4.6 shows where in the predictable `urandom` stream the SSH server read. We saw 27 different outputs during the 1,000 boots; the most frequent occurred 23.7% of the time.

Boot-time entropy hole The entropy sources we disabled would likely be missing in some headless and embedded systems, particularly on first boot. This means that there is a window of vulnerability—a boot-time entropy hole—during which Linux's `urandom` may

be entirely predictable, at least for single-core systems. If processes generate long-term cryptographic keys with entropy gathered during this window, those keys are likely to be vulnerable. The risk is particularly high for unattended systems that ship with preconfigured operating systems and generate SSH or TLS keys the first time the respective daemons start during the initial boot. Likewise, if processes maintain their own entropy pools which are seeded from `urandom` during this window and then maintained without the addition of further system entropy, any cryptographic randomness extracted from those pools is likely to be vulnerable.

On stock Ubuntu systems, these risks are somewhat mitigated: TLS keys must be generated manually, and OpenSSH host keys are generated during package installation, which is likely to be late in the install process, giving the system time to collect sufficient entropy. However, on the Fedora, Red Hat Enterprise Linux (RHEL), and CentOS Linux distributions, OpenSSH is installed by default, and host keys are generated on first boot. We experimented further with RHEL 5 and 6 to determine whether host keys on these systems might be compromised, and observed that sufficient entropy had been collected at the time of key generation (due to greater disk activity than with Ubuntu Server) by a slim margin. We believe that most server systems running these distributions are safe, particularly since they likely have multiple cores and gather additional entropy from physical concurrency. However, it is possible that other distributions and customized installations do not collect sufficient entropy on startup and generate weak keys on first boot.

4.5.2 Factorable RSA keys and OpenSSL

One interesting question raised by our vulnerability results is why factorable RSA keys occur at all. A naïve implementation of RSA key generation would simply seed a PRNG from the operating system's entropy pool and then use it to generate p and q. In this approach, we would expect to see duplicate keys if the OS provided the same seed, but factorable keys would be extremely unlikely. What we see instead is that some devices seem prone to generating keys with common factors. Another curious feature is that although some of the most common prime factors divided hundreds of different moduli, in nearly all of these cases the second prime factor did not divide any other keys.

One explanation for this pattern is that implementations updated their entropy pools in the middle of key generation. In this case, the entropy pool states might be identical as distinct key generation processes generate the first prime p and diverge while generating the second prime q.

In order to explore this theory, we studied the source code of OpenSSL [79], one of the most widely used open-source cryptographic libraries. OpenSSL is not the only software

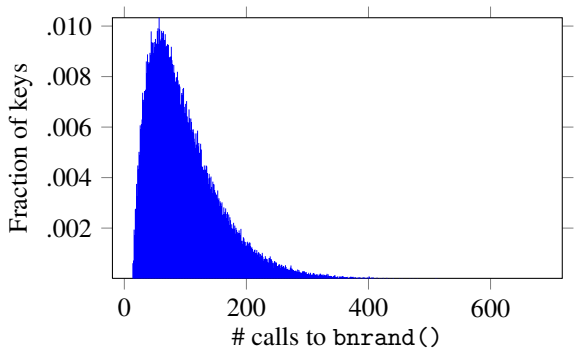


Figure 4.7: **Calls to bnrand() during key generation** — Every time OpenSSL extracts randomness from its internal entropy pool using `bnrand()`, it mixes the current time in seconds into the pool. This histogram shows the number of calls for 100,000 1024-bit RSA key generation operations. Both the average and the variance are relatively high due to the probabilistic procedure used to generate primes.

library responsible for the problems we observed, but we chose to examine it because the source code is freely available and because of its popularity. SSH version strings identified OpenSSH (which uses OpenSSL's key generation libraries) for 75% of SSH hosts and 93% of SSH hosts with factored RSA keys. For TLS, OpenSSL was identified in the `nsComments` field of a certificate for 0.6% of TLS hosts and 0.6% of TLS hosts with factored RSA keys. Mironov [181] observes that OpenSSL's prime generation code contains tests to generate "safe" primes (those for which $p-1$ has no small factors) and that this unusual property for prime factors can be used to identify implementations that use this code. We found that 3,861 of the 16,717 factored moduli (23%) satisfied this property, suggesting that 46 of the 54 device clusters of factorable keys we identified use OpenSSL or similar code for generating "safe" primes.

OpenSSL RSA key generation OpenSSL's built-in RSA key generator relies on an internal entropy pool maintained by OpenSSL. The entropy pool is seeded on first use with (on Linux) 32 bytes read from `/dev/urandom`, the process ID, user ID, and the current time in seconds. OpenSSL provides a function named `bnrand()` to generate cryptographically-sized integers from the entropy pool, which, on each call, mixes into the entropy pool the current time in seconds, the process ID, and the possibly uninitialized contents of a destination buffer. Note that, in identically configured systems, the only inputs to the entropy pool that should be expected to diverge are `urandom`, the time the pool is seeded, and the time of each `bnrand()` call.

The RSA key generation algorithm generates the primes p and q using a randomized algorithm. It selects a random integer using `bnrand()` and uses trial division to search for a candidate prime within a range. Once a candidate prime is found, it is passed through

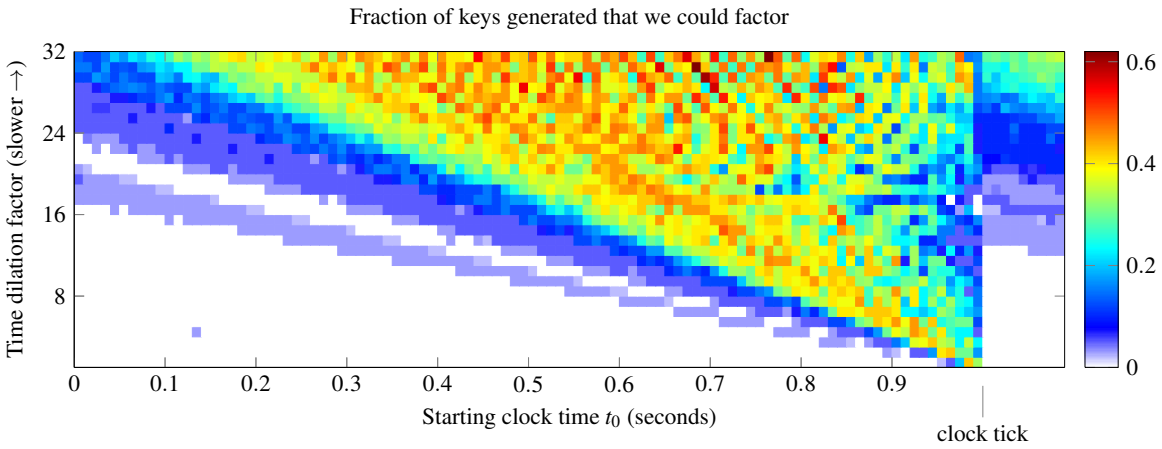


Figure 4.8: **OpenSSL generating factorable keys** — We hypothesized that OpenSSL can generate factorable keys under low-entropy conditions due to slight asynchronicity between the key generation process and the real-time clock, we generated 14 million RSA keys using controlled entropy sources for a range of starting clock times. Each square in the plot indicates the fraction of 100 generated keys that could we could factor. In many cases (white), keys are repeated but never share primes. After a sudden phase change, factorable keys occur during a range leading up to the second boundary, and that range increases as we simulate machines with slower execution speeds.

several rounds of the Miller-Rabin primality test, each of which also queries `bnrand()` to generate randomness. If the primality test fails, the above process is repeated with a new random integer until a likely prime is found.

During this process, OpenSSL extracts entropy from the entropy pool dozens to hundreds of times, as shown in Figure 4.7. Since we observed many keys with one prime factor in common, we can conjecture that multiple systems are starting with `urandom` and the time in the same state and that the entropy pool states diverge due to the addition of time during intermediate steps of the key generation process.

We hypothesized that, even when two executions begin with identical inputs to the entropy pool and identical clocks, the key generation process is hypersensitive to small variations in where the *boundary between seconds* falls. Slight variations in execution speed might cause the wall clock tick to fall between different calls to `bnrand()`, resulting in different execution paths. This can result in several different behaviors, with three simple cases:

If the second never changes while computing p and q, every execution will generate identical keys. If the clock ticks while generating p, both p and q diverge, yielding distinct keys with no shared factors. If instead the clock advances to the next second during the generation of the second prime q, then two executions will generate identical primes p but can generate distinct primes q based on exactly when the second changes.

Experiment To test our hypothesis, we modified OpenSSL 1.0.0g to control all the entropy inputs used during key generation, generated a large number of RSA keys, and determined how many were identical or factorable. We set the process ID, user ID, and uninitialized input buffer to constant values and modified the `time()` function to simulate the process starting at a configurable time t_0. Our experimental machine was a Dell Optiplex 980 with a 2.80 GHz Core i7 processor running Ubuntu Server 10.04.4. For $t_0 \in \{0, 0.01, \ldots, 1.10\}$ we generated 100 1024-bit RSA keys using each of 40 different hardcoded values in place of `/dev/urandom`. To simulate the effects of slower clock speeds, we dilated the clock time returned by `time()` and repeated the experiment using dilation multipliers of 1–32. In all, we generated 14 million keys. We checked for common factors within each batch of 100 keys.

The results we obtained, illustrated in Figure 4.8, are consistent with our hypothesis. At all time dilations, we find that factorable keys are generated during a range of start times leading up to the second boundary. No factorable keys are generated for low starting offsets, as both p and q are generated before the second changes. As the initial offset increases, there is a rapid phase change to generating factorable keys, as generation of q values begins to overlap the second boundary. Eventually, the fraction of factorable keys falls as the second boundary occurs during the generation of more p values, resulting in distinct moduli with no common factors.

The range of starting offsets when factorable keys occurred is narrow at fast clock speeds and spreads out as the time dilation factor increases and the simulated speed of the machine falls. For very slow simulated clocks, OpenSSL generates factorable keys for nearly the whole range of clock offsets. This may be reflective of the performance of many embedded devices.

While this theory can explain many of the device behaviors we observed, some devices produced distributions of factors that must have other causes. For example, the IBM RSA II remote administration devices (shown in Figure 4.3b) generated only nine distinct prime factors and the moduli appeared to be generated by selecting two of these prime factors uniformly at random. In this case, each modulus shared both of its prime factors with a large number of other devices. We saw evidence of this phenomenon only for a few hundred devices.

4.5.3 DSA signature weaknesses and Dropbear

The DSA signature vulnerabilities we observed indicate that entropy problems impact not only key generation but also the continued runtime behavior of server software during normal usage. This is somewhat surprising, since we might expect the operating system to

collect sufficient entropy *eventually*, even in embedded devices. We investigated Dropbear, a popular light-weight SSH implementation. It maintains its own entropy pools seeded from the operating system at launch, on Linux with 32 bytes read from `urandom`. This suggests a possible explanation for the observed problems: the operating system had not collected enough entropy when the SSH server started, and, from then on, even though the system entropy pool may have had further entropy, the running SSH daemon did not.

To better understand why these programs produce vulnerable DSA signatures, we examined the source code for the current version of Dropbear, 0.55. The ephemeral key is generated as output from an internal entropy pool. Whenever Dropbear extracts data from its entropy pool, it increments a static counter and hashes the result into the pool state. No additional randomness is added until the counter (a 32-bit integer) overflows, at which point the entropy pool is reseeded with data from `urandom`, the current time, and the process ID. This implies that, if two Dropbear servers are initially seeded with the same value from `urandom`, they will provide identical signature randomness as long as their counters remain synchronized and do not overflow.

(We note that Dropbear contains a routine to generate k in a manner dependent on the message to be signed, which would ensure that distinct messages are always signed with distinct k values and protect against the vulnerability that we explore here. However, that code is disabled by default.)

We looked for evidence of synchronized sequences of ephemeral keys in the wild by making further SSH requests to a handful of the Dropbear hosts from our scan. We chose two hosts with the SSH version string `dropbear-0.39` that had used identical DSA public keys and r values during our original SSH scan. There were 356 IP addresses in our dataset that had served that public key, and we rescanned these addresses to find any that might currently be synchronized. We received 291 responses, and one pair used identical r values. We immediately collected 50 further signatures from these two hosts, and found that the signatures followed an identical sequence of r values. We could advance the sequence of one host by making several SSH requests, then cause the other host to catch up by making the same number of requests. When probed again an hour later, both hosts remained in sync. This suggests that Dropbear is causing vulnerabilities in the manner we predicted.

Dropbear appeared in the version strings of 1,367,365 (13.4%) of the SSH hosts in our scans and 222 (5.09%) of the hosts with repeated DSA r values. Several other implementations, including hosts identifying OpenSSH and the Siemens Gigaset routers displayed similar behavior when rescanned. Because OpenSSL adds the current clock time to the entropy pool before extracting these random values, this suggests that some of these devices do not have a working clock at all.

4.6 Discussion

4.6.1 RSA vs. DSA in the face of low entropy

Any cryptosystem that relies on a secret key for security will be compromised if an adversary can determine that key. In practice, this might happen if an implementation leaks side-channel information about the key, or if the adversary can enumerate a reduced key space generated by low-entropy inputs. However, both RSA and DSA have specific catastrophic failure modes when implemented without sufficient entropy which can reveal the private key to an attacker who knows nothing about how the keys were generated. In a sense, the design of both cryptosystems permits the large-scale behavior of low-entropy implementations to act like a type of cryptographic side channel. From the point of view of robustness, these properties of the algorithms are less than ideal, but they result in vulnerability due to flawed implementations, not cryptographic flaws in either RSA or DSA.

We believe that the DSA signature vulnerabilities pose more cause for concern than the RSA factorization vulnerability. The RSA key factorization vulnerability that we investigated occurs only for certain patterns of key generation implementations in the presence of low entropy. In contrast, the DSA signature vulnerability can compromise any DSA private key—no matter how well generated—if there is ever insufficient entropy at the time the key is used for signing. It is not necessary to search for a collision, as we did; it suffices for an attacker to be able to guess the ephemeral private key k. The most analogous attacks against RSA of which we are aware show that some types of padding schemes can allow an attacker to discover the encrypted plaintext or forge signatures [60]. We are unaware of any attacks that use compromised RSA signatures to recover the private key.

We note that our findings show a larger fraction of SSH hosts are compromised by the DSA vulnerability than by factorable RSA keys, even though our scanning techniques have likely only revealed a small fraction of the hosts prone to repeating DSA signature randomness. In contrast, the factoring algorithm we used has found all of the repeated RSA primes in our sample of keys.

There are specific countermeasures that implementations can use to protect against these attacks. If both prime factors of an RSA modulus are generated from a PRNG without adding additional randomness during key generation, then low entropy would result in repeated but not factorable keys. These are more readily observable, but may be trickier to exploit, because they do not immediately reveal the private key to a remote attacker. To prevent DSA randomness collisions, the randomness for each signature can be generated as a function of the message and the pseudorandom input. (It is very important to use a cryptographically secure PRNG for this process [46].) Of course, the most important countermeasure is

for implementations to use sufficient entropy during cryptographic operations that require randomness, but defense-in-depth remains the prudent course.

4.6.2 /dev/(u)random as a usability failure

Linux and some other UNIX-like operating systems provide two interfaces for generating randomness: `/dev/random`, which blocks until the system estimates that its entropy pool contains enough fresh entropy to satisfy the request, and `/dev/urandom`, which immediately returns the requested number of bytes even if no entropy has been added to the pool. Neither adequately suits the needs of application developers.

The Linux documentation states that "[a]s a general rule, `urandom` should be used for everything except long-lived GPG/SSL/SSH keys" [4]. However, all the open-source implementations we examined used `urandom` to generate keys by default. Based on a survey of developer mailing lists and forums, it appears that this choice is motivated by two factors: `random`'s extremely conservative behavior and the mistaken perception that `urandom`'s output is secure enough.

As others have noted, Linux is very conservative at crediting randomness added to the entropy pool [125], and `random` further insists on using *freshly collected* randomness that has not already been mixed into the output PRNG. The blocking behavior means that applications that read from `random` can hang unpredictably, and, in a headless device without human input or disk entropy, there may never be enough input for a read to complete. While blocking is intended to be an indicator that the system is running low on entropy, `random` often blocks even though the system has collected more than enough entropy to produce cryptographically strong PRNG output—in a sense, `random` is often "crying wolf" when it blocks.

This behavior appears to have contributed to a widespread belief among developers that `random`'s blocking behavior is merely an annoyance to be worked around. A comment in the Dropbear source code reflects this view:

```
/* We'll use /dev/urandom by default, since /dev/random is too much
hassle.  If system developers aren't keeping seeds between boots
nor getting any entropy from somewhere it's their own fault.  */
#define DROPBEAR_RANDOM_DEV "/dev/urandom"
```

Our experiments suggest that many of the vulnerabilities we observed may be due to the output of `urandom` being used to seed entropy pools before any entropic inputs have been mixed in. Unfortunately, the existing interface to `urandom` gives the operating system no means of alerting applications to this dangerous condition. Our recommendation is that Linux should add a secure RNG that blocks until it has collected adequate seed entropy and

thereafter behaves like `urandom`.

4.6.3 Are we seeing only the tip of the iceberg?

Nearly all of the vulnerable hosts that we were able to identify were headless or embedded devices. This raises the question of whether the problems we found appear *only* in these types of devices, or if instead we are merely seeing the tip of a much larger iceberg.

Based on the experiments described in Section 4.5.1, we conjecture that there may exist further classes of vulnerable keys that were not visible to our methods, but could be compromised with targeted attacks. The first class is composed of embedded or headless devices with an accurate real-time clock. In these cases, keys generated during the boot-time entropy hole may appear distinct, but depend only on a configuration-specific state and the boot time. These keys would not appear vulnerable in our scanning, but an attacker may be able to enumerate some or all of such a reduced key space for targeted implementations.

A more speculative class of potential vulnerability consists of traditional PC systems that automatically generate cryptographic keys on first boot. We observed in Section 4.5.1 that an experimental machine running RHEL 5 and 6 did collect sufficient entropy in time for SSH key generation, but that the margin of safety was slim. It is conceivable that some lower-resource systems may be vulnerable.

Finally, our study was only able to detect vulnerable DSA ephemeral keys under specific circumstances where a large number of systems shared the same long-term key and were choosing ephemeral keys from the same small set. There may be a larger set of hosts using ephemeral keys that do not repeat across different systems but are nonetheless vulnerable to a targeted attack.

We found no evidence suggesting that RSA keys from standard implementations that were generated interactively or subsequent to initial boot are vulnerable.

4.6.4 Directions for future work

In this work, we examined keys from two cryptographic algorithms on two protocols visible via Internet-wide scans of two ports. A natural direction for future work is to broaden the scope of all of these choices. Entropy problems can also affect the choice of Diffie-Hellman key parameters and keying material for symmetric ciphers. In addition, there are many more subtle attacks against RSA, DSA, and ECDSA that we did not search for. We focused on keys, but one might also try to search for evidence of repeated randomness in initialization vectors in ciphertext or salts in cryptographic hashes.

We also focused solely on services visible to our scans of the public Internet. Similar vulnerabilities might be found by applying this methodology to keys or other cryptographic

data obtained from other resource-constrained devices that perform cryptographic operations, such as smart cards or mobile phones.

The observation that `urandom` can produce predictable output on some types of systems at boot may lead to attacks on other services that automatically begin at boot and depend on good randomness from the kernel. It warrants investigation to determine whether this behavior may undermine other security mechanisms such as address space layout randomization or TCP initial sequence numbers.

4.7 Defenses and Lessons

The vulnerabilities we have identified are a reminder that secure random number generation continues to be a challenging problem. There is a tendency for developers at each layer of the software stack to silently shift responsibility to other layers; a far better practice would be a defense-in-depth approach where developers at every layer apply careful security design and testing and make assumptions clear. We suggest defensive strategies and lessons for several important groups of stakeholders.

4.7.1 For OS developers:

Provide the RNG interface applications need. Typical security applications require a source of randomness that is guaranteed to be of high quality and has predictable performance; neither Linux's `/dev/random` nor `/dev/urandom` strikes this balance (see Section 4.6.2). The operating system should maintain a secure PRNG that refuses to return data until it has been seeded with a minimum amount of true randomness and is continually seeded with fresh entropy collected during operation.

Communicate entropy conditions to applications. The problem with `/dev/urandom` is that it can return data even before it has been seeded with any entropy. The OS should provide an interface to indicate how much entropy it has mixed into its PRNG, so that applications can gauge whether the state is sufficiently secure for their needs.

Test RNGs thoroughly on diverse platforms. Many of the entropy sources that Linux supports are not available on headless or embedded devices. These behaviors may not be apparent to OS developers unless they routinely test the internals of the entropy collection subsystem across the full spectrum on platforms the system supports.

4.7.2 For library developers:

Default to the most secure configuration. Both OpenSSL and Dropbear default to using `/dev/urandom` instead of `/dev/random`, and Dropbear defaults to using a less secure

DSA signature randomness technique even though a more secure technique is available as an option. In general, cryptographic libraries should default to using the most secure mechanisms available. If the library provides fallback options, the documentation should make the trade-offs clear.

Use RSA and DSA defensively. Crypto libraries can take specific steps to prevent weak entropy from resulting in the immediate leak of private keys due to co-factorable RSA moduli and repeated DSA signature randomness (see Section 4.6.1).

Heed warnings from the OS, and pass them on. If the OS provides a mechanism to signal that the entropy it has collected is insufficient for cryptographic use (such as blocking when `/dev/random` is read), do not proceed with key generation or other security-critical operations. Provide an interface to communicate these states to applications, and document the condition as security critical.

4.7.3 For application developers:

Generate keys on first use, not on install or first boot. When keys are generated on initial boot, little or no entropy may be available from the operating system. If keys must be generated automatically, it may be better to defer generation until the keys are needed (such as during the first SSH connection), by which point the system will have had more of a chance to collect entropy.

Heed warnings from below. If the OS or cryptography library being used raises a signal that insufficient entropy is available (such as blocking), applications should detect this signal and refuse to perform security-critical operations until the system recovers from this potentially vulnerable state. Developers have been known to work around low-entropy states by ignoring or disabling such warnings, with extremely dangerous results [124].

Frequently add entropy from the operating system. When using libraries that maintain their own entropy pools, periodically add fresh randomness from the operating system in order to take advantage of additional entropy collected by the system during use. This is especially important for long-running daemons that are typically launched at the boot, when little or no entropy may be available.

4.7.4 For device manufacturers:

Avoid factory-default keys or certificates. While some defense is better than nothing, default keys and certificates provide only minimal protection. A better approach is to generate fresh keys on the device after sufficient randomness has been collected or to force users to upload their own keys.

Seed entropy at the factory. Devices could be initialized with truly random seeds at the factory. Sometimes it is already necessary to configure unique state on the assembly line (such as to set MAC addresses), and entropy could be added at the same time.

Ensure entropy sources are effective. Embedded or headless devices may not have access to sources of randomness assumed by the operating system, such as user-input devices or disk timing. Device makers should ensure that effective entropy sources are present, and that these are being harvested in advance of cryptographic operations.

Test cryptographic randomness on the completed device. Device makers should not simply assume that standard OS functions or cryptographic libraries are functioning correctly. Rather, they should test the complete device to check for weak keys and other entropy-related failures. Device makers can repeat some of the entropy tests that we describe in this paper on a smaller scale by generating many keys under simulated usage conditions and checking for repeated keys, factorable moduli, and repeated signature randomness.

Use hardware random number generators when possible. Given the problems we have identified with entropy collection in software, security-critical devices should use a hardware random number generator for cryptographic randomness whenever possible. Simple, low-cost circuits can provide sufficient entropy to seed a PRNG [237], and Intel has started introducing this capability in new CPUs [235].

4.7.5 For certificate authorities:

Check for repeated, weak, and factorable keys Certificate authorities have a uniquely broad view of keys contained in TLS certificates. We recommend that they repeat our work against their certificate databases and take steps to protect their customers by alerting them to potentially weak keys.

4.7.6 For end users:

Regenerate default or automatically generated keys. Cryptographic keys and certificates that were shipped with the device or automatically generated at first boot should be manually regenerated. Ideally, certificates and keys should be generated on another device (such as a desktop system) with access to adequate entropy.

4.7.7 For security and crypto researchers:

Secure randomness remains unsolved in practice. The fact that all major operating systems now provide cryptographic RNGs might lead security experts to believe that any entropy problems that still occur are the fault of developers taking foolish shortcuts. Our findings

suggest otherwise: entropy-related vulnerabilities can result from complex interaction between hardware, operating systems, applications, and cryptographic primitives. We have yet to develop the engineering practices and principles necessary to make predictably secure use of unpredictable randomness across the diverse variety of systems where it is required.

Primitives should fail gracefully under weak entropy. Cryptographic primitives are usually designed to be secure under ideal conditions, but practice will subject them to conditions that are less than ideal. We find that RSA and DSA, with surprising frequency, are used in practice under weak entropy scenarios where, due to the design of these cryptosystems, the private keys are totally compromised. More attention is needed to ensure that future primitives degrade gracefully under likely failure modes such as this.

4.8 Related Work

4.8.1 HTTPS surveys

The HTTPS public-key infrastructure has been a focus of attention in recent years, and researchers have performed several large-scale scans to measure TLS usage and CA behavior. In contrast, our study addresses problems that are mostly separate from the CA ecosystem.

Ristic published an SSL survey in July 2010 [217] examining hosts serving the Alexa top 1 million domain names and 119 million other domain registrations. The study found 900,000 hosts serving HTTPS and 600,000 valid certificates. The same year, the Electronic Frontier Foundation (EFF) and iSEC Partners debuted the SSL Observatory project [102] and released the largest public repository of TLS certificates. They scanned approximately 87% of the IPv4 address space on port 443 and downloaded the resulting X.509 certificates over a three-month period. They released two datasets, the larger of which (from December 2010) recorded 4.0 million certificates from 7.7 million HTTPS hosts. The authors used their data to analyze the CA infrastructure and noted several vulnerabilities. We owe the inspiration for our work to their fascinating dataset, in which we first identified several of the entropy problems we describe; however, we ultimately performed our own scans to have more up-to-date and complete data. Our scan data contains approximately 67% more IP addresses and 45% more certificates than the latest publicly available SSL Observatory scan.

In 2011, Holz et al. [137] scanned the Alexa top 1 million domains and observed TLS sessions passing through the Munich Scientific Research Network (MWN). Their study recorded 960,000 certificates and was the largest academic study of TLS data at the time. They report many statistics gathered from their survey, mainly focusing on the state of the CA infrastructure. We note that they examined repeated keys and dismissed them as "curious, but not very frequent." Yilek et al. [254] performed daily scans of 50,000 TLS

servers over several months to track replacement time for certificates affected by the Debian weak key bug. Our count of Debian certificates provides another data point on this subject.

4.8.2 Problems with random number generation

Several significant vulnerabilities relating to weak random number generation have been found in widely used software. In 1996, the Netscape browser's SSL implementation was found to use fewer than a million possible seeds for its PRNG [114]. In May 2008, Bello discovered that the version of OpenSSL included in the Debian Linux distribution contained a bug that caused keys to be generated with only 15 bits of entropy [47]. The problem caused only 294,912 distinct keys to be generated per key size during a two year period before the error was found [254].In August 2010, the EFF SSL Observatory [102] reported that approximately 28,000 certificates, 500 of which were signed by browser-trusted certificate authorities, were still using these Debian weak keys.

Gutmann [124] draws lessons about secure software design from the example of developer responses to an OpenSSL update intended to ensure that the entropy pool was properly seeded before use. He observes that many developers responded by working around the safety checks in ways that supplied no randomness whatsoever. The root cause, according to Gutmann, was that the OpenSSL design left the difficult job of supplying sufficient entropy to library users. He concludes that PRNGs should handle entropy-gathering themselves.

Gutterman, Pinkas, and Reinmann analyzed the Linux random number generator in 2006 [125]. In contrast to our analysis, which focuses on empirical measurement of an instrumented Linux kernel, theirs was based mainly on a review of the LRNG design. They point out several weaknesses from a cryptographic perspective, some of which have since been remedied. In a brief experimental section, they observe that the only entropy source used by the OpenWRT Linux distribution was network interrupts.Dorrendorf, Gutterman, and Pinkas reverse-engineered the Windows random number generator [90] and discovered that an attacker with access to one state of the generator could enumerate all past and future states.

4.8.3 Weak entropy and cryptography

In 2004, Bauer and Laurie [43] computed the pairwise GCDs of 18,000 RSA keys from the PGP web of trust and discovered a pair with a common factor of 9, demonstrating that the keys had been generated with broken (or omitted) primality testing.

The DSA signature weakness we investigate is well known and appears to be folklore. In 2010, the hacking group fail0verflow computed the ECDSA private key used for code signing on the Sony PS3 after observing that the signatures used repeated ephemeral

keys [64]. Several more sophisticated attacks against DSA exist: Bellare, Goldwasser, and Miccancio [46] show that the private key is revealed if the ephemeral key is generated using a linear congruential generator, and Howgrave-Graham and Smart [139] give a method to compute the private key from a fraction of the bits of the ephemeral key.

Ristenpart and Yilek [216] developed "virtual machine reset" attacks in 2010 that induce repeated DSA ephemeral keys after a VM reset, and they implement "hedged" cryptography to protect against this type of randomness failure. Hedged public key encryption was introduced by Bellare et al. in 2009 and is designed to fail as gracefully as possible in the face of bad randomness [45].

As we were preparing this paper for submission, an independent group of researchers uploaded a preprint [162] reporting that they had computed the pairwise GCD of RSA moduli from the EFF SSL Observatory dataset and a database of PGP keys. Their work is concurrent and independent to our own; we were unaware of these authors' efforts before their work was made public. They declined to report the GCD computation method they used.

The authors of the concurrent work report similar results to our own on the fraction of keys that were able to be factored, and thus the two results provide validation for each other. In their paper, however, the authors draw very different conclusions than we do. They do not analyze the source of these entropy failures, and they conclude that RSA is "significantly riskier" than DSA. In contrast, we performed original scans that targeted SSH as well as TLS, and we looked for DSA repeated signature weaknesses as well as cofactorable RSA keys. We find that SSH DSA private keys are compromised by weak entropy at a higher rate than RSA keys, and we conclude that the fundamental problem is an implementation issue rather than a cryptographic one.

Furthermore, the authors of the concurrent work state that they "cannot explain the relative frequencies and appearance" of the weak keys they observed and report no attempt to determine their source. In this work, we performed extensive investigation to trace the vulnerable keys back to specific devices and software implementations, and we have notified the responsible developers and manufacturers. We examined the source code of several of these implementations and performed experiments to understand why entropy problems are occurring. We find that the weak keys can be explained by specific design and implementation failures at various levels of the software stack, and we make detailed recommendations to developers and users that we hope will lessen the occurrence of these problems in the future.

4.9 Conclusion

In this chapter, we investigated the security of random number generation on a broad scale by performing and analyzing the most comprehensive Internet-wide scans of TLS certificates and SSH host keys to date. Using the global view provided by Internet-wide scanning, we discovered that insecure RNGs are in widespread use, leading to a significant number of vulnerable RSA and DSA keys.

Our experiences suggest that a data-driven approach can help us identify subtle flaws in cryptographic implementations. Previous examples of random number generation flaws were found by painstakingly reverse engineering individual devices or implementations, or through luck when a collision was observed by a single user. Our scan data allowed us to essentially mine for vulnerabilities and detect problems in dozens of different devices and implementations in a single shot. Many of the collisions we found were too rare to ever have been observed by a single user but quickly became apparent with a near-global view of the universe of public keys.

More broadly, our experience highlighted how Internet-wide scanning can identify new classes of security weaknesses, but requires significant effort. This observation led us to develop ZMap, which we use in the other case studies in this dissertation.

CHAPTER 5

Analyzing the HTTPS Certificate Ecosystem

5.1 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 [102], and we collected 736% more unique certificates in total than any prior study of HTTPS [133].

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://httpsecosystem.org/.

5.2 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 [76] 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 [87]. 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.

5.3 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 [137]. 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 [102]. 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 [133]. Similarly, Yilek et al. performed daily scans of 50,000 TLS servers over several months to track the Debian weak key bug [254]. 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 [27]. 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.

5.4 Methodology

Our data collection 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.

5.4.1 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 [101]. 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 [101]. 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.

Scan	EFF [102]	Ps & Qs [133]	First	Representative	Latest	Total
Date Completed	2010-8	2011-10	2012-6-10	2013-3-22	2013-8-4	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 5.1: **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.

5.4.2 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 [174, 246]. 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 [190] 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.

5.4.3 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.

5.4.4 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. Similar to Holz et al. [137], 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 5.1. 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 5.2.

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 5.2: **Top 10 Countries Serving Trusted Certificates**

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 5.1). 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 of what was hosted at a single point in time, these short lived certificates would impact the breakdown of several of our statistics.

5.4.5 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 5.1. 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.

5.4.6 Server Name Indication Deployment

Both Holz et al. [137] and Akhawe et al. [27] 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 [55]. 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 as of Spring 2016 [189].

5.5 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 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.

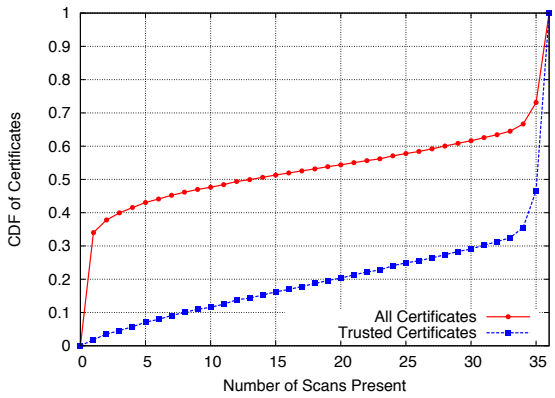


Figure 5.1: **CDF of Scan Presence by Certificate** — We performed 36 scans from 1/2013 to 3/2013. Here, we show the number of scans in which each certificate was found. We note that over 30% of self-signed certificates were only found in one scan.

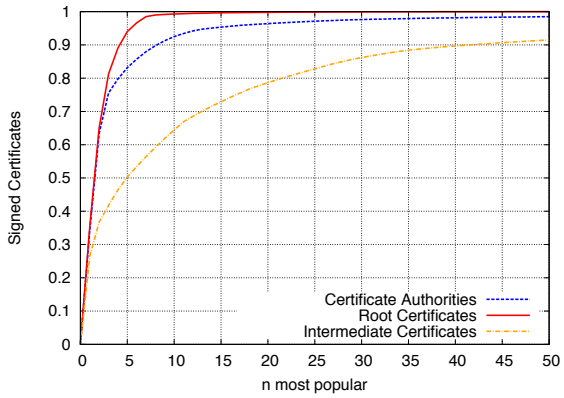


Figure 5.2: **CDF of Leaf Certificates by CA** — We find that 90% of trusted certificates are signed by 5 CAs, are descendants of 4 root certificates, and were signed by 40 intermediate certificates.

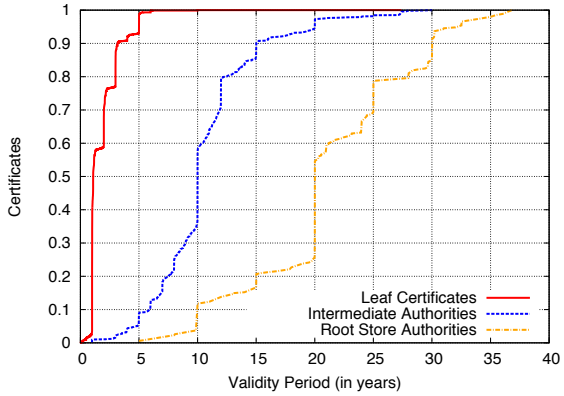


Figure 5.3: **Validity Periods of Browser Trusted Certificates** — Trusted CA certs are being issued with validity periods as long as 40 years, far beyond the predicted security of the keys they contain.

5.5.1 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 [217]. Holz et al. find 2,300 intermediate certificates in their active scanning [137]. 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 5.2. We classified the types of the organizations that control a CA certificate, which we show in Table 5.3. 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.

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	(94.2%)	3,260,454	(76.3%)
Government	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	(4.8%)
ISP	30	(4.37%)	58	(3.16%)	8,126	(0.23%)	8,126	(1.55%)
IT/Security	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	(14.7%)
Nonprofit Organization	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	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 5.3: **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 5.4: **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.

5.5.2 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 7.7, 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

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

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.

There is little inherent security risk associated with an organization controlling a CA certificate that is technically constrained to only allow signing certificates for that organization's set of domains. However, this is not the practice we observed. Rather, the signing certificates being provided to these organizations can sign for *any* domain, increasing the entire ecosystem's attack surface.

5.5.3 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 [102], 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 5.2, 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 5.4 and the top 10 commercial CAs in Table 5.5.

There is a long history of commercial CA compromise [52, 214, 230]. In each of these cases, web browsers and operating systems explicitly blacklisted the compromised signing certificate or misissued certificates [52, 192]. 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 [170]. As such, we would hope that large commercial authorities would distribute signing among a number of intermediate certificates. However, as seen in Figure 5.2, 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.

5.5.4 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 5.6, 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" [8]. It fails to mention that its certificates

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 5.6: **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.

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 [249].

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.

5.5.5 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 [138] 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 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.

5.5.6 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 [138] 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.

5.5.7 Authority Key Usage

All of the browser-trusted leaf certificates in our study were signed using an RSA key. As shown in Table 5.8, 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 [40]. However, as seen in Figure 5.5, more than 70% of CA certificates using 1024-bit keys expire after this date and

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 5.7: **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 5.8: **Key Distribution for Trusted Signing Certificates**

57% of roots using 1024-bit RSA keys have signed children that expire after 2016. Figure 5.3 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 5.4, 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.

5.6 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.

5.6.1 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 5.9. 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 [150]. While a large number of

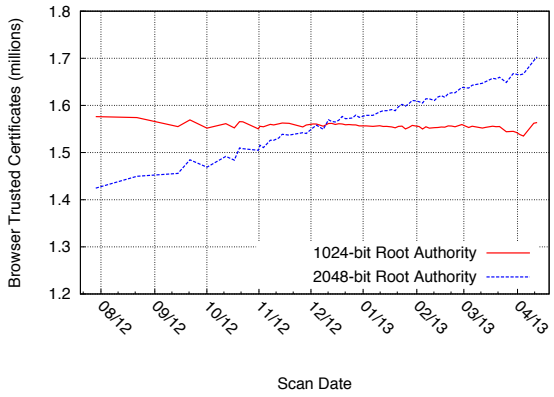


Figure 5.4: **Temporal Trends in Root Key Size** — We find that 48.7% of browser-trusted leaf certificates are dependent on 1024-bit RSA based root authorities, contrary to recommended practice [40].

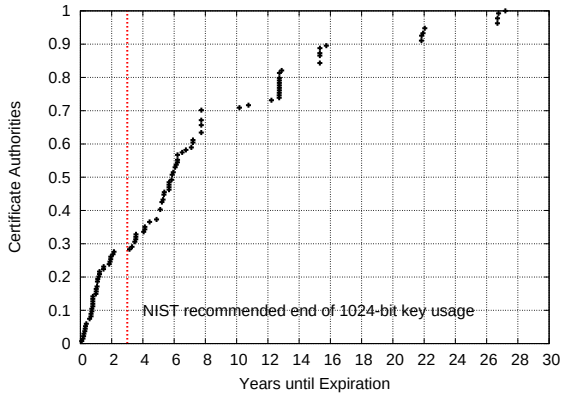


Figure 5.5: **Expiration of 1024-bit Root Certificates** — This figure shows when trusted 1024-bit RSA CA certificates expire. We note that more than 70% expire after 2016 when NIST recommends discontinuing the use of 1024-bit keys.

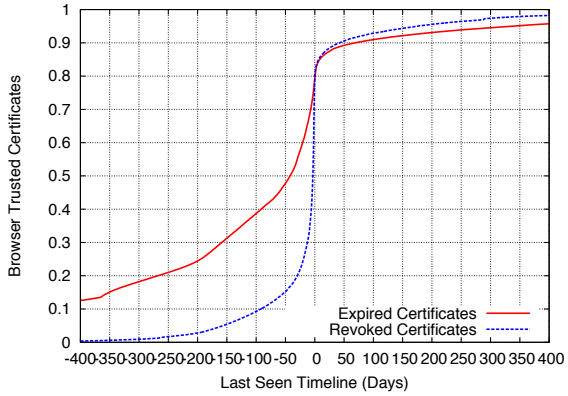


Figure 5.6: **CDF of Certificate Removal** — We find that 20% of expiring certificates and 19.5% of revoked certificates are removed retroactively (to the right of 0 days).

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 [133, 162, 254]. We revisit several of these measurements and provide up-to-date metrics. Following up on the study performed by Heninger et al. [133], 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 [47], of which 96 were browser trusted, a 34% decrease from 2011 [133]. The last browser-trusted certificate containing a Debian weak key was signed on January 25, 2012.

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 5.9: **Trusted Leaf Certificate Public Key Distribution**

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 5.10: **Trusted Leaf Certificate Signature Algorithms**

Signature Algorithms In line with the results presented by Holz et al. [137], 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" [230]. We provide a breakdown of leaf certificate signature types in Table 5.10.

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.

5.6.2 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 5.11. We note that Mozilla Network Security Services (NSS) [190], 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 5.6. 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.

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 5.11: **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.

5.6.3 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 [213].

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.

5.6.4 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 5.9. While RFC 5280 [76] 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 5.6, more than 80% of certificates are removed proactively and were not seen again after the time of their revocation.

WebTrust for Certificate Authorities [9], 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.

5.7 Unexpected Observations

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

5.7.1 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,

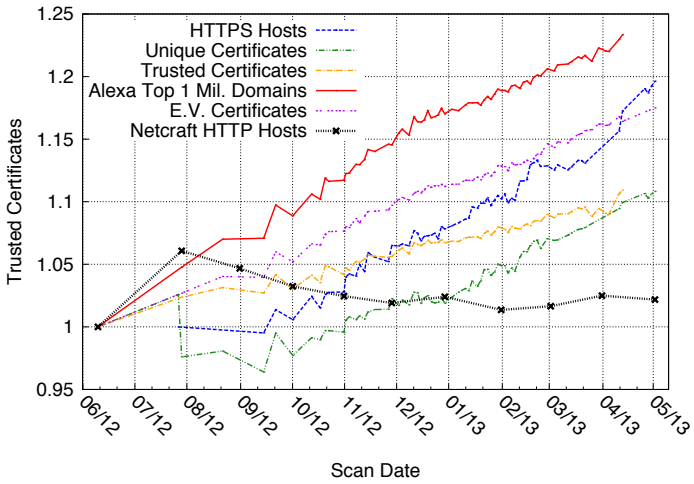


Figure 5.7: **Growth in HTTPS Usage** — Over the past 14 months, we observe between 10-25% growth of all aspects of HTTPS usage.

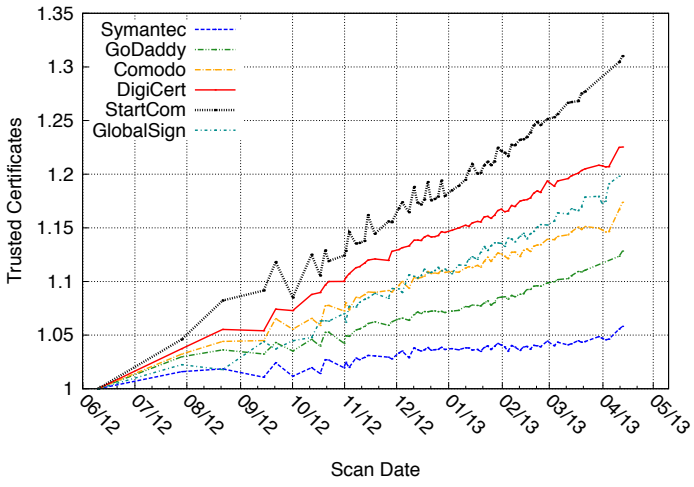


Figure 5.8: **Change in Authority Market Share** — In this figure, we should the individual growth of the top 10 most prolific certificate authorities.

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**	

Figure 5.9: **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 [52].

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).

5.7.2 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 [132], who factored the 512-bit key a few hours after the certificate expired.

5.7.3 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 [156] and was revoked by Turktrust on December 26, 2012. It was last seen in our scans on December 27, 2012.

5.7.4 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.

5.8 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 5.7. 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 [188]. 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 EV certificates, a 19.6% increase in the number of hosts serving HTTPS, and an 11.1% increase 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 5.8.

5.9 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*. 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. There are clear cases for using these restrictions, but the vast majority of the time, CAs are not 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 5.7.2). 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 5.7.3). 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 5.5.3).

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, it adopted guidelines for certificate authorities [65] 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 work group, 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 5.7.4).

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. We hope that our work can help provide the data needed for the forum to follow up with these organizations.

Browsers to Lead the Way Browser and OS 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, CAs lack incentive to adopt more secure options regardless of community support.

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 [147], 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 5.5). 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 [41]. 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 5.5.7). 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.

5.10 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.

CHAPTER 6

Uncovering Attacks on Email Delivery

6.1 Introduction

Electronic mail carries some of a user's most sensitive communication, including private correspondence, financial details, and password recovery confirmations that can be used to gain access to other critical resources. Users expect that messages are private and unforgeable. However, as originally conceived, SMTP—the protocol responsible for relaying messages between mail servers—does not authenticate senders or encrypt mail in transit. Instead, servers support these features through protocol extensions such as STARTTLS, SPF, DKIM, and DMARC. The impetus for mail servers to adopt these features is entirely voluntary. As a consequence, gradual rollout has led to a fractured landscape where mail servers must tolerate unprotected communication at the expense of user security. Equally problematic, users face a medium that fails to alert clients when messages traverse an insecure path and that lacks a mechanism to enforce strict transport security.

In this work, we measure the global adoption of SMTP security extensions and the resulting impact on end users. Our study draws from two unique perspectives: longitudinal SMTP connection logs spanning from January 2014 to April 2015 for Gmail, one of the world's largest mail providers; and a snapshot of SMTP server configurations from April 2015 for the Alexa Top Million domains. We use both perspectives to estimate the volume of messages and mail servers that support encryption and authentication, identify mail server configuration pitfalls that weaken security guarantees, and ultimately expose threats introduced by lax security policies that enable wide-scale surveillance and message forgery.

From Gmail's perspective, incoming messages protected by TLS have increased 82% over the last year, peaking at 60% of all inbound mail in April 2015. Outgoing messages similarly grew by 54%, with 80% of messages protected at the conclusion of our study in April. This improvement was largely fueled by a small number of popular web mail providers, including Yahoo and Outlook, enabling security features mid-year. However, such best practices continue to lag for the long tail of 700,000 SMTP servers associated with the

Alexa Top Million: only 82% support TLS, of which a mere 35% are properly configured to allow server authentication. We argue that low adoption stems in part from two of the three most popular SMTP software platforms failing to protect messages with TLS by default.

A similar split-picture emerges for the adoption of technologies such as SPF, DKIM, and DMARC that authenticate senders and guard against message spoofing. In terms of sheer volume, during April 2015, Gmail was able to validate 94% of inbound messages using a combination of DKIM (83%) and SPF (92%). However, among the Alexa Top Million mail servers, only 47% deploy SPF policies and only 1% provide a DMARC policy, the absence of which leaves recipients unsure whether an unsigned message is invalid or expected. When mail servers specify SPF policies, 29% are overly broad (covering tens of thousands of addresses.)

This security patchwork—paired with opportunistic encryption that favors failing open and transmitting messages in cleartext, so as to allow incremental adoption—enables network attackers to intercept and surveil mail. In one such attack, network appliances corrupt STARTTLS connection attempts and downgrade messages to non-encrypted channels. We identify 41,405 SMTP servers in 4,714 ASes and 193 countries that cannot protect mail from passive eavesdroppers due to STARTTLS corruption on the network. We analyze the mail sent to Gmail from these hosts and find that in seven countries, more than 20% of all messages are actively prevented from being encrypted. In the most severe case, 96% of messages sent from Tunisia to Gmail are downgraded to cleartext.

In the second attack, DNS servers provide fraudulent MX records for the SMTP servers of common mail providers. We searched for DNS servers that provide fraudulent addresses for Gmail's SMTP servers, and we find 14,600 publicly accessible DNS servers in 521 ASes and 69 countries. We investigate the messages that Gmail received from these hosts and find that in 193 countries more than 0.01% of messages from each country are transited through these impostor hosts. In the largest case, 0.08% of messages from Slovakia are relayed from a falsified IP, which can intercept or alter their contents.

Drawing on our measurements, we discuss various challenges and attacks, present current proposals for securing mail transport, and propose directions for future research. We hope that our findings can both motivate and inform further work to improve the state of mail security.

6.2 Background

Simple Mail Transfer Protocol (SMTP) is the Internet standard for sending and relaying email [151, 205]. Figure 6.1 illustrates a simplified scenario: a client sends mail by transmitting it to its local outgoing SMTP server, which relays each message to the incoming

SMTP server for the recipient's domain. In practice, mail forwarding, mailing lists, and other complications result in messages traversing multiple SMTP relays before arriving at their final destination.

As originally conceived in 1981, SMTP did not support protecting the confidentiality of messages in transit or authenticating messages upon receipt. Due to these shortcomings, passive observers can read message content on the wire, and active attackers can additionally alter or spoof messages. To address this gap in security, the mail community developed protocol extensions, such as STARTTLS, DKIM, DMARC, and SPF, to encrypt message content and authenticate senders.

6.2.1 Protecting Messages in Transit

STARTTLS is an SMTP extension introduced in 2002 that encapsulates SMTP within a TLS session [134]. In a typical STARTTLS session, a client first negotiates an SMTP connection with the server, after which the client sends the command `STARTTLS`, which initiates a standard TLS handshake. The client then transmits mail content, attachments, and any associated metadata over this cryptographically protected channel.

STARTTLS aims to protect the individual hops between SMTP servers, primarily protecting messages from passive eavesdroppers. As we will discuss in Section 6.4, STARTTLS is typically not used to authenticate destination mail servers, but rather provides opportunistic encryption. (This differs from the behavior of HTTPS clients, which strictly require TLS.) In almost all cases, mail servers do not validate presented certificates and will relay messages over cleartext if STARTTLS is not supported. Because STARTTLS only protects hops between individual relays, each relay still has access to messages and can freely read and modify message content.

The STARTTLS RFC [134] does not define how clients should validate presented certificates. While it suggests that the recipient's domain (e.g., `gmail.com`) should be present in the certificate, it also permits checking the fully qualified domain name (FQDN) of the MX server. This removes the need for third-party mail servers (e.g., shared hosting like Google Apps for Work) to present a trusted certificate for each hosted domain. However, it also enables network-level attackers to falsely report MX records that point to an attacker-controlled domain. Without additional security add-ons (e.g., DANE [93]), this attack remains a real threat.

6.2.2 Authenticating Mail

Mail servers deploy several complementary mechanisms for authenticating and verifying the integrity of received mail. While STARTTLS protects individual hops between servers,

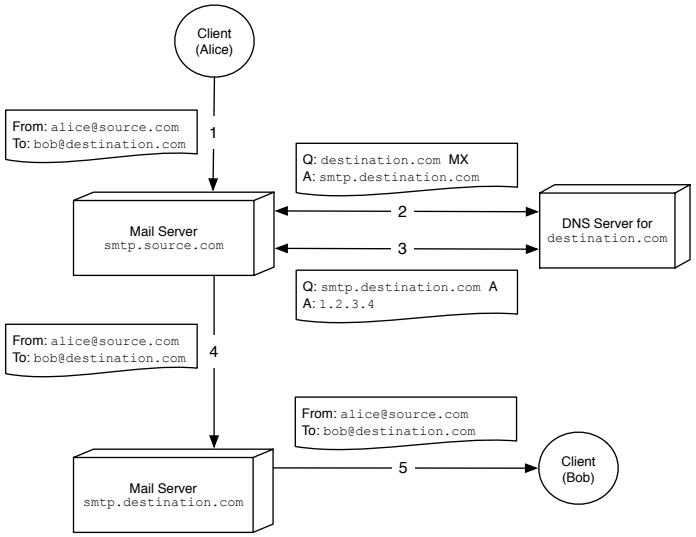


Figure 6.1: **SMTP Protocol**—A client sends outgoing mail by connecting to its organization's local SMTP server (❶). The local server performs a DNS lookup for the mail exchange (`MX`) record of the *destination.com* domain, which contains the hostname of the destination's SMTP server, in this case *smtp.destination.com* (❷). The sender's server then performs a second DNS lookup for the destination server's IP address (❸), establishes a connection, and relays the message (❹). The recipient can later retrieve the message using a secondary protocol such as POP3 or IMAP (❺).

these additional protocols allow recipients to verify that messages have not been spoofed or modified, and they provide a mechanism to report forged messages. A more detailed discussion of each protocol and its limitations is available from MAAWG [82]. We depict the interplay between these mechanisms in Figure 6.2.

DKIM DomainKeys Identified Mail (DKIM) lets SMTP servers detect whether a received message has been spoofed or modified during transit (RFC 6376 [81]). To utilize DKIM, a sender appends the `DKIM-Signature` field to the message header. This header contains a digital signature of the message signed with the private key tied to the sender's domain. Upon delivery, the recipient can retrieve the sender's public key through a DNS request and verify the message's signature. DKIM does not specify what action the recipient should take if it receives a message with an invalid or missing cryptographic signature. Instead, the organization must have a predetermined agreement with the sender.

SPF Sender Policy Framework (SPF) allows an organization to publish a range of hosts that are authorized to send mail for its domain (RFC 7208 [149]). To deploy SPF, the organization publishes a DNS record that specifies which hosts or CIDR blocks belong to the organization. Upon receiving mail, the recipient performs a DNS query to check for an SPF policy and can choose to reject messages that do not originate from the specified servers. SPF further allows organizations to delegate a portion or the entirety of their SPF

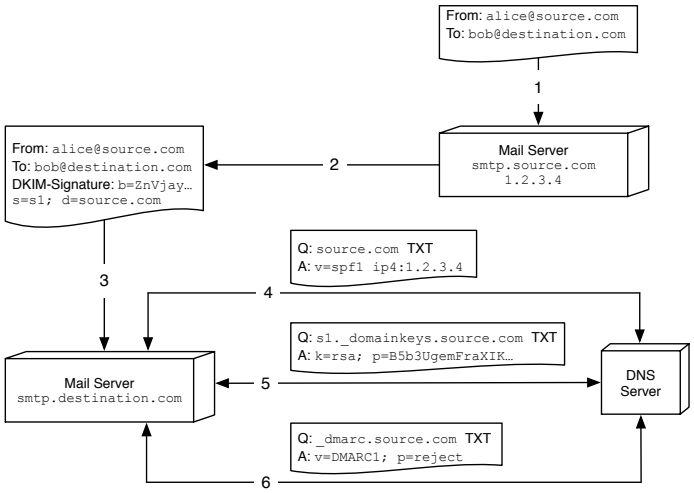


Figure 6.2: **Mail Authentication**—SPF, DKIM, and DMARC are used to provide source authentication. The outgoing server digitally signs the message (❷). The receiving mail server performs an SPF lookup (❹) to check if the outgoing server is whitelisted, a DKIM lookup (❺) to determine the public key used in the signature, and a DMARC lookup (❻) to determine the correct action should SPF or DKIM validation fail.

policy to another organization, and they commonly delegate SPF settings to a cloud provider (e.g., Google Apps for Work.)

DMARC Domain-based Message Authentication, Reporting, and Conformance (DMARC) builds upon DKIM and SPF and allows senders to suggest a policy for authenticating received mail (RFC 7489 [154]). Senders publish a DNS TXT record (named `_dmarc.domain.com`) that indicates whether the sender supports mail authentication (i.e., DKIM and/or SPF), and what action recipients should take if authentication fails (e.g., the DKIM signature is missing or invalid). DMARC further allows organizations to request daily reports on spoofed messages that other servers receive.

6.3 Dataset

Our study is based on two unique datasets: logs of the SMTP handshakes negotiated for mail sent to and from Gmail from January 2014 to April 2015, and a snapshot of SMTP server configurations from April 2015 for the Alexa Top Million domains.

Gmail Inbound and Outbound Messages Google publicly reports statistics about encrypted inbound and outbound messages via its Transparency Report.[1] This dataset explicitly excludes spam messages as not to conflate user security with bulk automated messages. We obtain a companion dataset via a collaboration with Google that contains the set of all ciphers negotiated with external SMTP servers during the same period, as well as any

[1] Data is available at http://www.google.com/transparencyreport/safermail.

Status	Top Million Domains	
No MX records	152,944	(15.29%)
No resolvable MX hostnames	5,447	(0.55%)
No responding SMTP servers	49,125	(4.91%)
SMTP Server	792,494	(79.25%)

Table 6.1: **Organizational SMTP Deployment**—We investigate how domains in the Alexa Top Million have deployed SMTP.

authentication performed on behalf of the sending party. We rely on this dataset for making observations about the volume of mail protected by encryption and authentication. We note that this dataset is noticeably skewed towards a handful of large web mail providers, including Yahoo and Outlook, as well as personal mail accounts provided by local ISPs that relay the bulk of the mail.

Alexa Top Million Mail Servers For a second perspective on organizational support for mail security, we examine the SMTP security features enabled by mail servers belonging to the Alexa Top Million ranked websites [1]. On April 26, 2015, we performed `MX` record lookups for the Alexa Top Million domains. For domains with mail servers, we followed up with a DNS query to identify supported SMTP security extensions (i.e., SPF and DMARC) and attempted an SMTP and STARTTLS handshake using ZMap [96, 101] to identify whether the mail servers support encryption.

In total, 792,494 domains (79.2% of the Alexa Top Million) have operational mail servers, as detailed in Table 6.1. The remaining domains include sites such as `t.co`, `googleusercontent.com`, and `blogspot.com` that do not need incoming mail servers.

6.4 Confidentiality in Practice

We measure the state of mail confidentiality based on the volume of messages protected by STARTTLS that are sent and received by Gmail, and the fraction of mail servers that support and correctly configure encryption.

6.4.1 Gmail

As of April 26, 2015, Gmail successfully initiated STARTTLS connections for 80% of outgoing messages, while 60% of incoming connections initiated a STARTTLS session. This represents a 54% increase (52% to 80%) in outbound and an 82% increase (33% to 60%) in inbound connections that utilize STARTTLS since January 2014. While this growth is encouraging, overall gains arrived in bursts rather than consistent growth, as shown in Figure 6.3. We point out two periods of immediate interest. Between May 10 and 30, 2014, outbound encryption jumped from 47% to 71%. This was likely due to Yahoo and Outlook

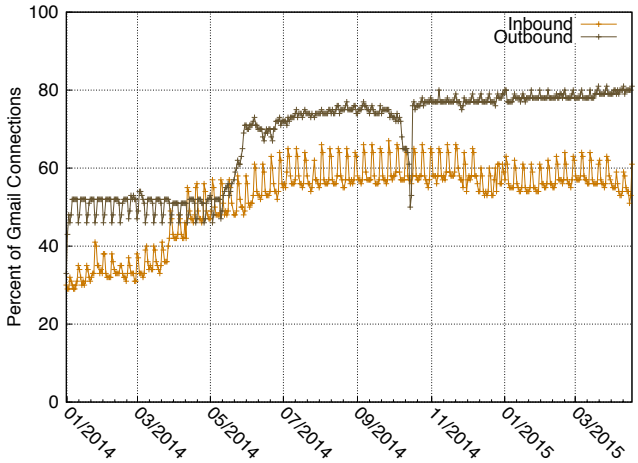


Figure 6.3: **Historical Gmail STARTTLS Support**—Inbound connections that utilize STARTTLS increased from 33% to 60% for weekdays between January 2014 and April 2015. Weekends consistently have close to 10% more connections that support STARTTLS than weekdays. Support for outgoing STARTTLS increased from 52% to 80% during this period.

deploying STARTTLS. Second, between October 8 and 17, outbound STARTTLS dropped from 73% to a low of 50%. The lowest point occurred on October 14, which corresponds with the public disclosure of the POODLE vulnerability [95]. We suspect the correlated drop was a result of mail server misconfigurations introduced by administrators attempting to disable SSLv3.

Influence of Major Email Providers Major mail providers, such as Gmail, Yahoo, and Outlook, heavily skew the apparent adoption of STARTTLS in contrast to the long tail of organizations that run their own mail servers. Of the 877 most common domains that Gmail transited mail to on April 26, 2015, only 58% accepted 100% of messages over TLS. Similarly, only 29% of 26,406 inbound mail domains encrypted 100% of messages. We explore this skew further in Section 6.4.2 from the perspective of the Top Million domains.

Along these same lines, we argue that the periodicity present in Figure 6.3 stems from users sending less business mail on weekends and instead relying on personal accounts provided by major providers. In particular, on weekdays between April 1 and 26, 2015, Gmail encrypted 79.8% of outbound messages, while mail servers encrypted 53.7% of incoming connections. Weekends during this same period saw an average 7.2% increase in the number of secured messages. As we discuss in Section 6.4.4, major mail providers support encryption by default.

Negotiated Cipher Suites We analyzed the cipher suites chosen by incoming Gmail connections on April 30, 2015, and found that 84.2% of TLS connections (45.2% of all

TLS Version	Key Exchange	Symmetric Cipher	HMAC	Inbound Traffic
TLSv1.2	ECDHE	AES-128-GCM	SHA-256	51.500%
TLSv1	ECDHE	RC4	SHA-1	29.225%
TLSv1	RSA	RC4	SHA-1	14.403%
TLSv1.2	ECDHE	AES-128	SHA-1	1.586%
TLSv1.2	RSA	RC4	SHA-1	1.147%
TLSv1	ECDHE	AES-128	SHA-1	0.999%
TLSv1.1	ECDHE	RC4	SHA-1	0.723%
TLSv1.2	RSA	AES-128-GCM	SHA-256	0.203%
SSLv3	RSA	RC4	SHA-1	0.060%
TLSv1.2	ECDHE	RC4	SHA-1	0.060%
TLSv1	RSA	AES-128	SHA-1	0.050%
TLSv1.1	RSA	RC4	SHA-1	0.024%
TLSv1.1	ECDHE	AES-128	SHA-1	0.011%
TLSv1.1	ECDHE	AES-256	SHA-1	0.004%
TLSv1.2	RSA	AES-256	SHA-1	0.003%
TLSv1.2	RSA	AES-128	SHA-1	0.001%
TLSv1	RSA	RC4	MD5	0.001%

Table 6.2: **Cipher Suites for Inbound Gmail Traffic**—80% of inbound Gmail connections are protected by TLS. Here, we present the selected cipher suites for April 30, 2015.

incoming connections) chose a perfect forward secret cipher suite. However, similar to HTTPS, 45.63% of clients continue to prefer RC4 despite its known weaknesses [30, 243]. For the remaining connections, 95% utilized AES-128-GCM and 5% used AES-128 (Table 6.2). We note that while this perspective does not show any known-broken ciphers (e.g., EXPORT suites), these may still occur between other mail servers. These do not appear in our dataset because Google does not support these ciphers, and if a client does not support any modern ciphers the TLS handshake will fail.

Comparison With Prior Estimates We compare Gmail's perspective with prior estimates published by Facebook [107]. During May 2014, Facebook—which sends mail notifications for friend requests and new user activity—successfully encrypted 58% of outbound messages. Of the mail servers contacted, 76% supported STARTTLS. By August 2014, Facebook successfully encrypted 95% of outbound notifications after several large webmail providers, notably Microsoft and Yahoo, deployed STARTTLS.

During this same time period, outbound encrypted messages for Gmail increased from 47% to 74%. Despite the same opportunistic STARTTLS policy, we find Gmail generally has a lower percentage of outgoing mail protected by STARTTLS than Facebook. We believe this stems from Facebook primarily communicating with personal mail accounts provided by major providers (e.g., Gmail, Yahoo, and Outlook) as opposed to businesses. We note that our Gmail measurements may have similar but less pronounced biases towards

Status	Top Million Domains	
SMTP Server—No STARTTLS support	144,464	(18.2%)
SMTP Server—STARTTLS support	648,030	(81.8%)

Table 6.3: **STARTTLS Deployment by Top Million Domains**—Our scan results show that 79% of Alex Top Million domains have incoming SMTP servers, of which 81.8% support STARTTLS.

MailProvider	Domains	STARTTLS	Trusted Certificate	Certificate Matches
Gmail	126,419 (15.9%)	Yes	Yes	server
GoDaddy	36,229 (4.6%)	Yes	Yes	server
Yandex	12,326 (1.6%)	Yes	Yes	server
QQ	11,295 (1.4%)	Yes	Yes	server
OVH	8,508 (1.1%)	Yes	Yes	mismatch
Other	597,717 (75.4%)	–	–	–

Table 6.4: **Top Mail Providers for Alexa Top Million Domains**—Five providers are used for mail transport by 25% of the Top Million domains. All five support STARTTLS for incoming mail.

large providers, which we investigate further in the next section.

6.4.2 Organizational Deployment

Given the skew present in Gmail's message volume towards major mail providers, we provide an alternate perspective by analyzing STARTTLS support for the 792,494 Alexa Top Million domains that advertise mail servers. In total, 648,030 (81.8%) of mail-enabled domains supported STARTTLS, as shown in Table 6.3. Only 5 domains within the Alexa Top 50 did not: `wikipedia.org`, `vk.com`, `weibo.com`, `yahoo.co.jp`, and `360.cn`. Support for encrypted mail was bolstered in part by 25% of domains outsourcing their mail servers to common third-party providers, such as Gmail, GoDaddy, Yandex, QQ, and OVH, all of which support STARTTLS. We give more details about these providers and their popularity in Table 6.4.

	Matches Domain	Matches Server	Matches Neither
Trusted	4,602 (0.6%)	270,723 (34.2%)	143,113 (18.1%)
Untrusted	4,345 (0.6%)	21,057 (2.7%)	181,242 (22.9%)
Total	8,947 (1.1%)	291,780 (36.8%)	324,355 (41.0%)

Table 6.5: **Certificates for Top Million Domains**—While 52% of domains' SMTP servers present trusted certificates, only 34.2% of trusted certificates match the MX server, and only 0.6% are valid for the recipient domain.

Key and Cipher Suites With the exception of two mail servers, all present certificates with RSA keys: 10.0% of domains use 1024-bit keys, 86.4% use 2048-bit keys, and 3% use 4096-bit or larger keys. Only 316 domains present 512-bit RSA certificates (which provide little to no security in 2015 [132]). We found 25.3% of domains support perfect forward secrecy and completed an ephemeral Diffie-Hellman key exchange. In addition, 59.2% of domains use RC4 and 40.8% use AES; only 25 domains select 3DES. In summary, most domains that deploy STARTTLS also deploy secure certificates. However, as in the HTTPS ecosystem, domains are slow in deploying modern, secure cipher suites.

Certificate Validity As part of the STARTTLS handshake, each mail server presents an X.509 certificate. While RFC 3207 [134] suggests that certificates match the mail domain (e.g., `gmail.com`), it also permits certificates that only match the name of the server in the domain's MX record (e.g. `aspmx.l.google.com`). However, certificates that match the MX server do not provide true authentication unless the MX records for the domain are cryptographically signed. Otherwise, an active attacker can return the names of alternate, attacker-controlled MX servers in the initial MX query. In practice, DNSSEC has not been widely deployed—recent studies have found that less than 0.6% of `.com` and `.net` domains have deployed DNSSEC [244]—and so operators cannot rely on this protection.

In our scan, 414,374 Top Million domains (52% of domains with valid SMTP servers, and 64% of domains that support STARTTLS) present certificates that validate against the Mozilla NSS root store [190], as detailed in Table 6.5. However, only 0.6% of domains present trusted certificates that match their domain name, while 34.2% present trusted certificates that match their MX server.

Surprisingly, 18.1% of domains present trusted certificates that match neither. These are primarily due to several mail hosting providers, including `psmtp.com` and `pphosted.com`, that incorrectly deployed wildcard certificates. In the remaining cases, certificates were simply for different domains. Another 33,281 domains present expired certificates, 60 certificates are signed by unknown CAs, and 55 certificates are invalidly signed by a parent certificate whose type mismatched the child certificate.

In summary, the certificates used by mail servers are in disarray. Less than 35% of mail servers with STARTTLS can be authenticated in any form, and a sender can only confirm that their connection had not been intercepted by an active attacker for 0.6% of domains. This has likely occurred because, as we discuss in the next two sections, common SMTP implementations and popular mail providers do not validate certificates, so server operators have little incentive to purchase and maintain a certificate.

6.4.3 Common Software Implementations

In order to understand why such a large number of organizations have not deployed STARTTLS and why only half of inbound connections to Gmail initiate a STARTTLS connection, we investigated the five most popular SMTP implementations, which account for 97% of identifiable mail servers for the Top Million domains. We tested whether each implementation initiated STARTTLS connections, whether it supported STARTTLS for incoming connections, and how it validated certificates. We installed the latest version of each SMTP server on an Ubuntu 14.04.1 LTS system, except for Microsoft Exchange, which was readily documented online [179]. The results are summarized in Table 6.6.

By default, Microsoft Exchange, Exim, and Sendmail initiate STARTTLS connections when delivering messages. Postfix and qmail—which together account for nearly 35% of all identifiable mail servers on the public IPv4 address space—send all messages over cleartext unless explicitly configured to use STARTTLS. All of the servers we tested fail open and send mail in cleartext if STARTTLS is not available.

Postfix and Microsoft Exchange Server support inbound STARTTLS connections by default by generating a self-signed certificate upon installation. This provides immediate protection against passive attacks without user configuration. The remaining servers do not accept TLS connections without manual configuration. Postfix and Exchange—the two servers that have confidentiality protection enabled by default—account for 22% of servers associated with Top Million domains.

Postfix was the only server capable of performing both server-based and domain-based certificate validation, although its documentation specifically recommends *against* enabling validation when interacting with the greater Internet [206]. Exim, qmail, Sendmail, and Microsoft Exchange do not support validating the destination domain when relaying mail.

6.4.4 Popular Mail Providers

Since a large fraction of mail is transited through a small number of popular providers, a single change can have a large impact on the entire ecosystem, as previously demonstrated in Figure 6.3. We measured inbound and outbound STARTTLS support for 19 common webmail providers and Internet service providers. We created an account on each provider and then sent mail to a Postfix server that we configured to support STARTTLS with a self-signed certificate. To test incoming STARTTLS support, we connected to the mail servers listed in each domain's MX record and initiated a STARTTLS handshake.

We summarize our results in Table 6.7. Only one provider, Lycos, did not support inbound STARTTLS. Two providers—facebookmail and OVH—presented certificates that

Mail Software	Top1M Share	IPv4 Share	STARTTLS Incoming	STARTTLS Outgoing	Server Validation	Domain Validation	Reject Invalid
exim 4.82	34%	24%	◐	●	○	○	○
Postfix 2.11.0	18%	21%	●	◐	◐	◐	◐
qmail 1.06	6%	1%	◐	◐	○	○	○
sendmail 8.14.4	5%	4%	◐	●	○	○	○
Exchange 2013	4%	12%	●	●	◐	○	◐
Other	3%	<1%					
Unknown	30%	38%	● default behavior | ◐ supported but not default | ○ no support				

Table 6.6: **Popular Mail Transfer Agents (MTA)**—We investigated the default behavior for five popular MTAs. By default, Postfix and qmail do not initiate STARTTLS connections. All five MTAs we tested fail open to cleartext if the STARTTLS connection fails.

matched neither their domain nor the hostname of their MX server. *None* of the providers presented a certificate that matched their domain, and thus none could have provided strong authentication in the presence of an active attacker who falsified the service's MX records.

Less than half of the providers negotiated perfect-forward-secret cipher suites. When sending mail, three of the providers—Lycos, GoDaddy, and OVH—did not initiate STARTTLS connections. The remaining providers initiated STARTTLS connections but did not validate certificates; in effect, this provides opportunistic encryption but no authentication.

Provider	Incoming TLS Version	Incoming Key Exchange	Incoming Cipher	Certificate Matches	Outgoing TLS Version	Outgoing Key Exchange	Outgoing Cipher
Gmail	1.2	ECDHE	AES-128-GCM	server	1.2	ECDHE	AES-128-GCM
Yahoo	1.2	ECDHE	AES-128-GCM	server	1.0	ECDHE	RC4-128
Outlook	1.2	ECDHE	AES-256-CBC	server	1.2	ECHDE	AES-256
iCloud	1.2	ECDHE	AES-128-GCM	server	1.2	DHE	AES-128-GCM
Hushmail	1.2	RSA	RC4-128	server	1.2	ECDHE	AES-256-GCM
Lycos	–	–	–	–	–	–	–
Mail.com	1.2	ECHDE	AES-256-CBC	server	1.2	DHE	AES-256-GCM
Zoho	1.0	RSA	RC4-128	server	1.0	RSA	RC4-128
Mail.ru	1.2	RSA	RC4-128	server	1.2	ECDHE	AES-256-GCM
AOL	1.0	RSA	RC4-128	server	1.0	DHE	AES-256-CBC
QQ	1.1	RSA	RC4-128	server	1.0	DHE	AES-256-CBC
Me.com	1.2	ECHDE	AES-128-GCM	server	1.2	DHE	AES-128-GCM
facebookmail	1.0	RSA	AES-128-CBC	mismatch	1.0	ECDHE	AES-128
GoDaddy	1.2	RSA	RC4-128	server	–	–	–
Yandex	1.2	RSA	AES-128-GCM	server	1.2	ECDHE	AES-256-CBC
OVH	1.2	RSA	AES-128-GCM	mismatch	–	–	–
Comcast	1.2	RSA	RC4-128	server	1.2	DHE	AES-128-CBC
AT&T	1.2	ECDHE	AES-128-GCM	server	1.0	ECDHE	RC4-128
Verizon	1.2	RSA	AES-128-GCM	server	1.0	DHE	AES-128-CBC

Table 6.7: **Encryption Behavior of Mail Providers**—We measured support for incoming and outgoing STARTTLS among various popular mail providers. While most providers supported STARTTLS, *none of them* validated our certificate, which was self-signed.

Provider	Servers Providing Invalid MX Answers	Servers Providing Invalid IP Answers	Unique Invalid MX Servers	Unique Invalid IPs	Responsive Invalid Mail Servers
Gmail	30,931	23,134	146	1,150	144
Yahoo	31,219	55,459	130	1,117	114
Outlook.com	29,618	23,145	117	1,059	110
Mail.ru	31,214	25,796	97	1,053	110
QQ	30,091	55,467	122	1,171	111

Table 6.8: **Fraudulent DNS Responses**—We scanned the public IPv4 address space for DNS servers that returned falsified MX records or SMTP server IP addresses for five popular mail providers. This data excludes loopback addresses and obvious configuration errors.

	Nov. 2013	Apr. 2015	Change
Overall failure rate	10.65%	6.14%	−4.42%
Crypto failures:			
Weak crypto key (<1024 bits)	21.00%	15.08%	−5.92%
Key is revoked	0.02%	0.01%	−0.01%
Signature algorithm not supported	0.27%	0.26%	−0.02%
Key is expired		0.06%	
Body hash doesn't match signature		18.66%	
Protocol version incorrect	0.59%	3.32%	+2.73%
Some DKIM tags are duplicated		0.05%	
Other error	77.91%	62.55%	−15.36%

Table 6.9: **Gmail DKIM Errors**—We present the breakdown of Gmail DKIM validation failures for November 2013 and April 2015.

6.4.5 Takeaways

Our results show that there has been significant growth in STARTTLS adoption over the past year. However, much of this growth can be attributed to a handful of large providers. In contrast, as seen in our scans, smaller organizations continue to lag in deploying STARTTLS, and as of March 2015 nearly half of inbound weekday connections remain unencrypted. This may be due, in part, to several popular implementations failing to initiate STARTTLS connections by default.

In the cases where encryption is present, messages are protected opportunistically. Connections fail open to cleartext if any issues arise during the handshake or if STARTTLS is not supported. None of the popular providers and implementations we tested use TLS for authentication, and only one common implementation supports validating a certificate against the destination domain.

Unfortunately, in the protocol's current form, mail providers cannot fail closed in the absence of STARTTLS until there is near total deployment of the extension, and until organizations deploy valid certificates, relays will be unable to automatically authenticate destination servers.

6.5 Threats to Confidentiality

As deployed in practice, STARTTLS protects connections against passive eavesdroppers but does not protect against active man-in-the-middle attacks. We examine two types of network attacks that this enables—downgrading STARTTLS sessions to insecure channels and falsifying MX records to re-route messages—and measure the prevalence of both methods in the wild.

Scan Result	IPv4 Hosts
TCP port 25 open	14,131,936
Responsive SMTP server	8,850,664
Successful STARTTLS handshake	4,620,561

Table 6.10: **IPv4 SMTP Scan Results**—We could perform a STARTTLS handshake with 52% of the SMTP servers that our IPv4 scans identified.

Category	IPv4 Hosts	
Command not echoed	3,606,468	(85.26%)
`STARTTLS` echoed correctly	617,093	(14.59%)
`STARTTLS` replaced	5,756	(0.14%)
Command truncated to four characters	786	(0.02%)

Table 6.11: **Detecting STARTTLS Manipulation**—We could extract an echoed command from 14.75% of servers that sent errors in response to our `STARTTLS` command. 0.14% of these responses indicate that the command was tampered with before reaching the server.

Type	ASes	
Corporation	182	(43.0%)
ISP	74	(17.5%)
Financial	57	(13.5%)
Academic	35	(8.3%)
Government	30	(7.1%)
Healthcare	14	(3.3%)
Unknown	12	(2.8%)
Airport	9	(2.1%)
Hosting	7	(1.7%)
NGO	3	(0.7%)

Table 6.12: **ASes Stripping STARTTLS**—We categorize the 423 ASes for which 100% of SMTP servers showed behavior consistent with STARTTLS stripping.

	Top Million Domains	IPv4 Hosts
Cisco-style tampering	2,563	41,405
`BLUF` tampering	0	6

Table 6.13: **Styles of STARTTLS Stripping**—The most prominent style of manipulation matches the advertised behavior of Cisco security devices and affects 41K SMTP servers.

Category	IPv4 Hosts
DNS servers	13,766,099
Responsive DNS servers	8,860,639
Any invalid MX responses	234,756
Class of invalid behavior:	
Identical response regardless of request	131,898
Returns loopback address	16,015
Returns private network address	7,680
Flipped bits in response	56,317
Falsified DNS record	178,439

Table 6.14: **Invalid or Falsified MX Records**—We scanned the IPv4 address space for DNS servers that provided incorrect entries for the MX servers for five popular mail providers.

6.5.1 STARTTLS Corruption

An active attacker—or a legitimate organization with a vested interest in snooping mail—can prevent mail encryption by tampering with the establishment of a TLS session. In this attack, a network actor takes advantage of the fail-open design of STARTTLS—where SMTP servers fall back to cleartext if any errors occur during the STARTTLS handshake—to launch a downgrade attack. A network actor can manipulate packets containing the `STARTTLS` command to prevent mail servers from establishing a secure channel, or alter a mail server's `EHLO` response to remove `STARTTLS` from the list of server capabilities. To measure whether STARTTLS sessions are being downgraded in the wild, we attempted to initiate STARTTLS connections with SMTP servers throughout the public IPv4 space and looked for evidence of tampering.

Scanning Methodology To find servers where STARTTLS is blocked, we build on the fact that SMTP servers frequently report back invalid commands they receive—which would include any corrupted `STARTTLS` command. We performed a TCP SYN scan of the public IPv4 address space on port 25 and attempted to perform an SMTP and STARTTLS handshake with responsive hosts. We performed this scan on April 20, 2015, from the University of Michigan campus using ZMap [101].

We found 14.1M hosts with port 25 open, 8.9M SMTP servers, and 4.6M SMTP servers that supported STARTTLS (see Table 6.10). Of the 4.2M hosts that failed to complete a TLS handshake, 623,635 (14%) echoed back the command they received. We classify these responses in Table 6.11. 617,093 (98.95%) of the responding hosts returned `STARTTLS` (and indeed do not to support it), 5,750 (0.92%) returned `XXXXXXXX`, 786 (0.14%) responded with `STAR` or `TTLS`, and 6 responded with `BLUF`.

The `STAR` and `TTLS` commands are four-character command truncations and are likely

not due to an attack. Prior to ESMTP, SMTP commands were all four characters, and we were able to confirm that all commands were truncated to four characters on these servers. However, the XXXXXXXX and BLUF commands appear due to the STARTTLS command being altered to prevent the establishment of a TLS session.

Affected Servers Excluding four-character truncations, our scan found 5,756 servers that display evidence of a corrupted STARTTLS command. However, given that only 14% of servers reported back the received command, this is likely an underestimate. We extended our search for servers where the STARTTLS command was corrupted in the server's list of advertised features, which is returned in response to the EHLO command. This identified an additional 35,649 servers. When combined with the initial set, this yields a total of 41,405 servers that apparently have STARTTLS messages corrupted. These 41K servers are located in 4,714 ASes (15% of all ASes with an SMTP server) and 191 countries (86% of countries with SMTP servers). They transit mail for 2,563 domains in the Top Million.

In 423 ASes (736 hosts), 100% of SMTP servers are affected by STARTTLS stripping. The AS performing stripping on 100% of the inbound and outbound mail with the most SMTP servers (21) belongs to Starwood Hotels and Resorts (AS 13401). We show the classification of ASes with 100% stripping in Table 6.12. Overall, no single demographic stands out; the distribution is spread over networks owned by governments, Internet service providers, corporations, and financial, academic, and health care institutions. We note that several airports and airlines appear on the list, including an AS belonging to a subsidiary of Boingo (AS 10245), a common provider of in-flight and airport WiFi.

Our scanning methodology does not comprehensively find all servers where STARTTLS is blocked. Local SMTP servers may not be accessible from the University of Michigan, STARTTLS might only be stripped for outgoing messages, or the command might be removed altogether instead of being corrupted in place. However, it appears that the practice is widespread.

Possible Causes The XXXXXXXX replacements are likely caused by security products that intercept and strip the command. In one prominent example, Cisco Adaptive Security Appliances (ASA) [71] and Cisco IOS Firewalls [72] both advertise replacing the STARTTLS command with Xs to facilitate mail inspection as part of their *inspect smtp* and *inspect esmtp* configurations. By inspecting messages, Cisco advertises that their products are capable of searching for and dropping messages with invalid characters in mail addresses, invalid SMTP commands, and long commands that may be attempting to exploit buffer overflows [73]. Table 6.13 shows the prominence of this style of tampering. While Cisco advertises this functionality, we cannot necessarily attribute every instance seen in the wild to Cisco devices, since others could implement stripping the same way.

Tunisia	96.13%	Reunion	9.28%
Iraq	25.61%	Belize	7.65%
Papua New Guinea	25.00%	Uzbekistan	6.93%
Nepal	24.29%	Bosnia and Herzegovina	6.50%
Kenya	24.13%	Togo	5.45%
Uganda	23.28%	Barbados	5.28%
Lesotho	20.25%	Swaziland	4.62%
Sierra Leone	13.41%	Denmark	3.69%
New Caledonia	10.13%	Nigeria	3.64%
Zambia	9.98%	Serbia	3.11%

Table 6.15: **Countries Affected by STARTTLS Stripping**—We measure the fraction of incoming Gmail messages that originate from the IPs that we found were stripping TLS from SMTP connections. Here, we show the countries with the most mail affected by STARTTLS stripping and the affected percentage of each country's incoming mail between April 20 and 27, 2015.

We are unable to attribute the `BLUF` replacement to any commonly known security software. The six hosts affected by this replacement also had the `PIPELINING` and `CHUNKING` capabilities in the `EHLO` response masked to `HIPELINING` and `PHUNKING`, respectively. Only those six hosts displayed this behavior; all were located in Ukraine.

Impact on Transited Mail To understand the volume of mail affected by STARTTLS corruption, we measure the number of messages transited to/from these devices from Gmail's perspective. The overall fraction of mail affected is small, but a handful of countries have a high local stripping rate (see Table 6.15). In the most extreme example, 96.13% of mail transited from Tunisia to Gmail is affected by STARTTLS stripping. Another 8 countries experience over 10% stripping, and 16 experience more than 5% stripping.

It is important to note that the devices that are stripping TLS from SMTP connections are not inherently malicious, and many of these devices may be deployed to facilitate legitimate filtering. Regardless of the intent, this technique results in messages being sent in cleartext over the public Internet, enabling passive eavesdropping and other attacks. Furthermore, the Cisco documentation does not discuss the downsides of enabling this functionality, and administrators may not be aware that the setting puts users at risk. Instead of stripping TLS, manufacturers should consider deploying in-line devices that accept and initiate STARTTLS connections, allowing them to inspect messages before securely forwarding them.

6.5.2 DNS Hijacking

Mail security, like that of many other protocols, is intrinsically tangled with the security of DNS resolution. Rather than target the SMTP protocol, an active network attacker can spoof the DNS records of a destination mail server to redirect SMTP connections to a server under

Slovakia	0.08%
Romania	0.04%
Bulgaria	0.03%
India	0.02%
Israel	0.01%
Switzerland	0.01%
Poland	0.01%
Ukraine	0.01%

Table 6.16: **Countries Affected by Falsified DNS Records**—We measure the fraction of mail received by Gmail on May 21, 2015 from IP addresses pointed to by false Gmail DNS entries. Here, we show the breakdown of mail from each country that originates from one of these addresses for the countries with the most affected mail.

the attacker's control. We investigated the prevalence of DNS servers that provide false MX records or SMTP server IP addresses for: gmail.com, yahoo.com, outlook.com, qq.com (a popular Chinese webmail provider), and mail.ru (a popular Russian webmail provider). We find evidence that 178,439 out of 8,860,639 (2.01%) publicly accessible DNS servers provided invalid IPs or MX records for one or more of these domains (see Table 6.8).

Scanning Methodology We identified servers responding with falsified DNS records by scanning the IPv4 address space ten times with ZMap in search of open resolvers and subsequently requesting the MX and A record of gmail.com, yahoo.com, outlook.com, qq.com, and mail.ru. We performed these scans on April 25, 2015, from the University of Michigan. In total, we identify 13.8 million DNS servers, of which 8.9 million resolved at least one query and 235K provide an invalid or falsified MX record (see Table 6.14).

We then performed a secondary scan, in which we repeated the same queries as well as performed A record lookups for a domain unrelated to mail transit (umich.edu) and a nonexistent domain. Of the 235K servers that provided invalid responses in our first scan, 56K supplied correct results in the secondary scan and were incorrectly flagged due to erroneous bit flips. Excluding those hosts, 132K provide the same publicly accessible address for every DNS query regardless of the domain, 7.7K provide reserved or private addresses, 16K respond with a loopback address, and 17.2K did not appear to spoof answers but were missing at least one of the MX servers. The devices that provided identical responses to every query or were missing an MX server appeared to be improperly configured embedded devices rather than malicious. After removing these hosts, we were left with 14.6K hosts that provided invalid responses for mail servers. These hosts pointed to 1,150 unique falsified mail servers, of which 144 (12.5%) completed an SMTP handshake.

Our scans do not provide an exhaustive list of hosts that might be intercepting mail. Since open resolvers are frequently used to launch DDoS attacks, most recursive DNS

resolvers are not externally accessible and will not appear in our scans. Similarly, many of the addresses we recorded are private, non-routable addresses, so we are unable to test whether mail transits through these hosts. However, our scan still finds upwards of 15K open resolvers that provide fraudulent responses when queried about mail providers and 1.2K false mail servers, which allows us to determine whether mail server DNS hijacking occurs in the wild.

Responsible Networks The DNS servers that provide fraudulent responses are located in 521 ASes. 83.6% of the hosts were located in five ASes: 62% Unified Layer (American Hosting Provider), 11.7% ChinaNet, 5.3% Telecom Italia (Italian ISP), 2.4% SoftLayer Technologies, and 2.0% eNom. In the case of Unified Layer, 9K hosts point back to seven servers, of which two completed SMTP handshakes. The hosts in the ChinaNet AS point to a range of loopback and private addresses and to 42 publicly routable servers, of which one completed an SMTP handshake. The devices in the Telecom Italia AS returned monotonically increasing IP addresses within 198.18.1.0/24 for all queries and do not appear to be specifically intercepting mail queries. The SoftLayer hosts respond with one of 18 addresses, of which 10 were SMTP servers. All eNom hosts point to a single IP, which did not accept SMTP connections. The remaining 2,386 DNS servers are located in 533 ASes and in 69 countries.

Impact on Transited Mail While a number of servers appear to be intercepting mail, the impact on transited messages is unclear. We estimate the amount of mail affected by measuring the number of inbound messages that Gmail received from each of the servers. The vast majority of mail that transits from these hosts is spam, but a small number of non-spam messages are sent through these servers. As shown in Table 6.16, in the most extreme case, upwards of 0.08% of mail transited from Slovakia came from falsified servers.

Whether malicious or well-intentioned, STARTTLS stripping and falsified DNS records highlight the weakness inherent in the fail-open nature and lack of authentication of the STARTTLS protocol. These attacks are both readily found in the wild and pose a real threat to users, with more than 20% of mail being sent in cleartext within seven countries.

6.6 Authentication in Practice

While STARTTLS protects messages against passive eavesdropping, it does not provide authentication. Mail can be modified or spoofed altogether, even in the presence of STARTTLS. As described in Section 6.2, SPF, DKIM, and DMARC have been developed to authenticate incoming mail. In this section, we measure how these protocols have been deployed in practice. We summarize the deployment of all three protocols in Table 6.18.

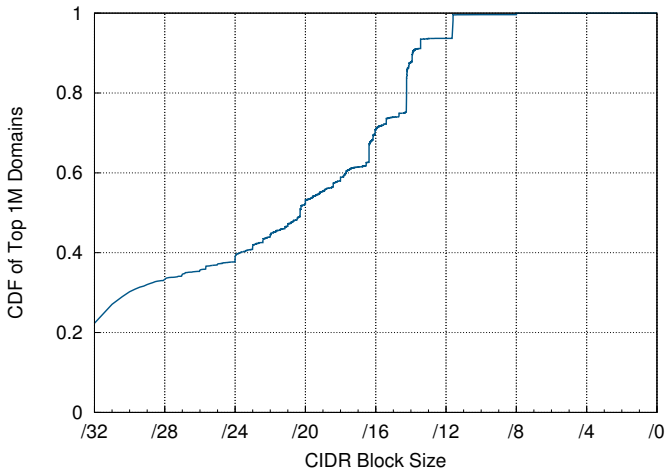


Figure 6.4: **Size of SPF Permitted Networks**—We show the CDF of the number of addresses whitelisted in a recursive resolution of the SPF records for Top Million domains.

6.6.1 SPF

When supported, SPF allows recipients to check that incoming messages from a domain (e.g., gmail.com) originate from an IP range authorized by that organization. From Gmail's perspective, Google successfully authenticated 92% of inbound messages during April 2015 using SPF, as detailed in Table 6.19. Of the unauthenticated messages, Gmail fails to validate 0.42% due to failures fetching the domain's SPF record; all other messages come from domains without an SPF policy.

Similar to STARTTLS deployment, the servers associated with the Top Million domains show significantly slower adoption with only 401,356 domains—47% of the Top Million domains with MX records—publishing an SPF policy (see Table 6.20). Of those, 255,867 domains allow mail to be sent if the server has an MX record on the domain, and 104 domains allow mail from hosts with reverse DNS names that match the domain.

Record Delegation Of the Top Million domains, 10,432 redirect (or fully delegate their SPF policy) to another provider, and 213,464 (53.2%) include records from one or more other domains' SPF policies. While this could potentially open an attack vector if multiple organizations specified the same IP blocks, we find that this was not the case. Instead, domains commonly include records from or redirect to a few well-known cloud mail providers. In the case of redirection, 35.7% delegate to Yandex (a Russian mail provider), 16.7% to mailhostbox.com, 8.0% to nicmail.ru, 3.9% to serveriai.lt, and 3.5% to mail.ru. 3,813 (36.6%) of all redirects point to a Russian mail service. For includes, 136,473 domains (64% of domains with includes) include one of five large mail providers: Gmail (59,660), Outlook (44,216), websitewelcome.com (20,291), mandrillapp.com (16,606), and SendGrid

Provider	SPF Policy	DMARC Policy
Gmail	soft fail	none
Yahoo	neutral	reject
Outlook	soft fail	none
Live	soft fail	–
iCloud	soft fail	none
Hushmail	soft fail	–
Lycos	soft fail	–
Mail.com	fail	–
Zoho	soft fail	–
Mail.ru	soft fail	none

Provider	SPF Policy	DMARC Policy
AOL	soft fail	reject
QQ	soft fail	none
Me.com	soft fail	none
Facebook	fail	reject
GoDaddy	fail	none
Yandex	soft fail	–
OVH	neutral	–
Comcast	neutral	none
AT&T	–	–
Verizon	neutral	–

Table 6.17: **SPF and DMARC Policies**—The majority of popular mail providers we tested posted an SPF record, but only three used the "strict fail" policy. Even fewer providers posted a DMARC policy, of which only three used "strict reject."

(10,700).

SPF Policy Coarseness For the domains that specify SPF policies, we find evidence that some report overly broad IP ranges that potentially enable an attacker to spoof mail origins, as shown in Figure 6.4. We find that 133,490 domains (60.9%) specify SPF CIDR ranges larger than a /24, 99,698 (29.2%) specify CIDR ranges larger than or equal to a /16, and 1,333 (0.4%) specify more than a /8. The vast majority of the domains that include a /8 mistakenly allowed messages from 10.0.0.0/8. In rarer cases, we find evidence of blatant misconfiguration: 62 domains specify network ranges akin to 255.255.255.255/8, and 20 domains specify ranges larger than a /8.

Policy Types Of the domains that deploy SPF, 21.7% adopt hard-failure policies, where recipients should reject mail from outside of the specified network. Another 58.0% adopt soft-failure policies, where recipients should accept the mail but consider it suspect (e.g., mark as spam), and 20.3% set no policy. If we restrict our analysis to the top mail providers, we find that most publish SPF records with a soft-fail policy (see Table 6.17). Exceptions include Facebook Mail, Mail.com, and GoDaddy, all of which have hard-fail policies. AT&T was the only provider we checked that does not have a valid SPF policy. We note that soft-fail policies allow more leeway for the destination domain to decide how to process a message, such as considering other spam indicators instead of downright rejecting messages.

6.6.2 DKIM

As a complement or alternative to SPF, DKIM allows a recipient to confirm the integrity and authenticity of inbound messages, even in the presence of a man-in-the-middle attack. In April 2015, 83.0% of the messages that Gmail received contained a DKIM signature. Of

Authentication Method	Nov. 2013	Apr. 2015	Change
DKIM & SPF	74.66%	81.01%	+6.31%
DKIM only	2.25%	1.98%	−0.27%
SPF only	14.44%	11.41%	−2.99%
No authentication	8.65%	5.60%	−3.00%

Table 6.18: **Gmail Incoming Mail Authentication**—During April 2015, 94.40% of incoming Gmail messages were authenticated with DKIM, SPF, or both.

SPF Policy	Gmail Messages
DNS timeout	<0.001%
Temporary error	0.184%
Permanent error	0.141%
Invalid record	0.098%

Table 6.19: **SPF Errors for Incoming Gmail Traffic**—We show the breakdown of errors fetching SPF records for incoming mail. Temporary errors can be fixed by retrying later; permanent errors mean the record was unable to be fetched.

the signed messages, 6.14% failed to validate due to weak cryptographic keys, revoked keys, or protocol errors (see Table 6.9). This represents a 4.42% decrease in invalid signatures when compared to two years prior. We specifically call attention to the fact that 18.7% of failures in April 2015 arose due to DKIM signatures not matching a message's content. While we cannot distinguish between malice and misconfiguration, such failures reflect the importance of authentication to alert mail servers to potential tampering, and they emphasize the imperative for the remaining 17% of unauthenticated inbound Gmail messages to adopt DKIM signatures. Unfortunately, due to the nature of the DKIM protocol, we cannot directly measure how many domains in the Alexa Top Million have deployed DKIM.

6.6.3 DMARC

DMARC policies allow sending mail servers to alert recipients that they support DKIM and SPF and then inform recipients how to handle incoming messages that fail to validate or

Policy	Top Million Domains	Recursive Top Million
SPF policy	401,356	401,356
Hard fail	84,801 (21.13%)	86,919 (21.65%)
Soft fail	226,117 (56.34%)	232,736 (57.99%)
Neutral	80,394 (20.03%)	81,701 (20.36%)
Redirect	10,045 (2.50%)	0 (0.00%)

Table 6.20: **SPF Policies for Top Million Domains**—We queried the SPF policies for the Top Million domains for both the top-level record and for full recursive resolution.

Record Type	Top Million Domains	Recursive Top Million
IPv4	200,976 (33.08%)	344,844 (40.22%)
IPv6	6,862 (1.13%)	108,086 (12.61%)
A	139,979 (23.04%)	148,688 (17.34%)
MX	249,345 (41.04%)	255,867 (29.84%)
REDIRECT	10,432 (1.72%)	0 (0.00%)

Table 6.21: **SPF Record Types for Top Million Domains**—We show how hosts are whitelisted within an SPF record for both the top-level SPF record and for full recursive resolution.

Published Policy	Gmail Messages	Top Million Domains
Quarantine	1.34%	709 (0.09%)
Empty	11.66%	6,461 (0.82%)
Reject	13.08%	1,720 (0.22%)
Not published	73.92%	783,851 (98.9%)

Table 6.22: **DMARC Policies**—We categorize DMARC policies for incoming Gmail messages from April 2015 and for Top Million domains with MX records on April 26, 2015.

that lack a DKIM signature. In contrast to the 90% of the messages that Gmail can validate with SPF or DKIM, only 26.1% of all inbound Gmail messages come from domains with a published DMARC policy. This discrepancy limits the effectiveness of DKIM as, absent a publicized policy, recipients cannot determine whether the lack of a signature is intended or is an indication of spoofing. For those messages with a policy, we provide a detailed breakdown in Table 6.22. We found that the majority of policies favored rejection (13%), though a significant fraction did not specify any action (11%).

A similarly pessimistic picture emerges for the Alexa Top Million, for which only 1.1% of domains with MX records published DMARC policies. We provide a breakdown of all the policies in Table 6.22 and the policies of top mail providers in Table 6.17. Even when DMARC is present, the majority of Alexa domains and even the top mail providers specified an empty policy. Only Yahoo, AOL, and Facebook advertised DMARC reject policies.

We suspect that many organizations have yet to deploy DMARC due to its relatively young age—RFC 7489 was first introduced in March 2013 and was last updated in March 2015, one month prior to our measurements. However, we note its necessity in enforcing SPF and DKIM policies, and we hope it will see increased deployment moving forward.

6.7 Discussion

The mail community has retroactively applied several security measures to SMTP. Nearly 60% of incoming connections to Gmail are encrypted and 94% of messages are authenticated

with DKIM or SPF. In many ways, this is a feat, given that SMTP did not originally provide any support for transport security. However, our two perspectives paint drastically different pictures of how mail security has been deployed. As can be seen by the 51% jump in encrypted inbound messages when Microsoft and Yahoo deployed STARTTLS, much of this success can be attributed to large mail providers that are pushing security forward. Unfortunately, as our scans demonstrate, smaller organizations lag in deploying security mechanisms correctly. While mail delivery security is rapidly improving, there are several structural problems that the mail community must address to guarantee the confidentiality and integrity of mail.

6.7.1 Challenges for Confidentiality

There are several challenges to guaranteeing the confidentiality of mail in transit. First, unlike HTTPS, there is no mechanism in SMTP for servers to indicate that mail should be protected by TLS. Further, users cannot indicate that mail should only be transited securely nor can they detect whether a message traversed a secure path.

In HTTPS, users can choose not to communicate over an insecure channel, and HSTS allows sites to indicate that future connections must use HTTPS. However, in SMTP, messages are relayed in cleartext if TLS cannot be negotiated. As we showed in Section 6.5.1, this has led to organizations corrupting the STARTTLS negotiation to force mail to be sent in the clear. Whether this is being done for legitimate or nefarious purposes, it illustrates that STARTTLS provides no protection against frequently occurring man-in-the-middle (MITM) attacks.

Second, even when TLS is used, there is no robust way for a sender to verify the authenticity of the recipient mail server. Common MTAs can validate that a server's certificate matches the destination domain's MX record, but not the destination domain name itself. Unfortunately, this still leaves the server open to impersonation unless the DNS responses are separately authenticated. As we showed in Section 6.5.2, certain entities are using this weakness to redirect the flow of messages.

One potential option for preventing MITM attacks is to create a mechanism similar to HTTP Public Key Pinning [104] for SMTP. This would allow a mail server to indicate whether future connections should require TLS and specify a public key. Other protections being adopted for the HTTPS PKI might also be considered for STARTTLS, such as the use of Certificate Transparency [157] to guard against dishonest or compromised certificate authorities.

Finally, we note that end-to-end mail encryption, as provided by PGP [66] and S/MIME [211], does not address many of the challenges we discuss in this work. While these solutions do

safeguard message content, they leave metadata, such as the subject, sender, and recipient, visible everywhere along the message's path. This information is potentially exposed to network-based attackers due to the lack of robust confidentiality protections for SMTP message transport. Although greater adoption of end-to-end encryption would undoubtedly be beneficial, for now, the overwhelming majority of messages depend solely on SMTP and its extensions for protection.

6.7.2 Challenges for Integrity

A major open question surrounding mail integrity is how to authenticate mail sent through mailing lists. Mailing lists frequently modify messages in transit, and DKIM signatures are invalidated by these modifications, which prevents large mail providers from publishing a DMARC reject policy. When Yahoo deployed a reject policy in 2014, it resulted in a heavy number of complaints and service malfunctions [75].

A second challenge is ensuring strong integrity as organizations move to cloud providers, where mail infrastructure and IP address blocks may be shared with other organizations. This infrastructure sharing is challenging in two respects. First, SPF has become less relevant, since, as explained in Section 6.6.1, SPF records tend to be overly broad. Second, DKIM becomes threatened by massive key compromises, as was the case for the SendGrid leak [67]. Overall, these two issues are part of a larger open question: How do we reliably establish the legitimacy of senders—whether for spam prevention or for integrity purposes—when many senders, good and bad, share common infrastructure?

The issue of shared infrastructure also affects mail confidentiality, as third-party providers would need to have certificates containing their clients' domains to allow strict certificate verification. This is problematic, as it opens the door to attacks where the third-party mail provider—or an attacker who breaches their systems—uses these certificates to impersonate the clients' domains, either for mail delivery or for HTTPS connections. This threat might be mitigated with a scope-reducing X.509 extension or through some other mechanism not yet devised.

6.8 Related Work

There has been little formal measurement of the public key infrastructure that supports mail transport until recently. There are three works similar to ours.

The first is a set of Facebook blog posts [106, 107] that describe the STARTTLS configurations seen from the perspective of Facebook notifications. In May 2014, Facebook found that 28.6% of notification emails are transported over a STARTTLS connection with strict certificate validation, 28.1% are protected with opportunistic encryption (indicating a

misconfigured STARTTLS server), and 41.0% of notifications are sent in cleartext. In August 2014, Facebook posted follow-up statistics in which they note that 95% of notification emails are sent over STARTTLS with strict certificate validation. Facebook further notes that this rise is primarily due to two major mail providers, Yahoo and Microsoft, deploying STARTTLS. The jump of encrypted messages from 28.6% to 95% is incredibly exciting. However, as noted by Facebook, their notification emails are skewed towards personal addresses and large hosting providers, such as Gmail and Yahoo Mail.

Concurrently with our work, Foster et al. [109] performed a similar study on the deployment of SMTP extensions for the Top Million domains present in the 2013 Adobe data breach [36] (ranked by number of accounts) and investigated how different types of senders (e.g., popular banks and e-commerce sites) protect mail. They found that 89% of popular mail providers deploy STARTTLS, 85% have SPF records, and 68% have DMARC policies. In comparison, at the termination of our study in April 2015, 54% of incoming messages to Gmail were protected by STARTTLS, and 82% of the domains to which Gmail transited mail supported inbound STARTTLS. While protocol deployment appears higher in their work, this falls in line with the trends we see: popular providers have deployed security extensions more comprehensively than smaller organizations.

In June 2014, Sean Rijs [215] published a measurement study on the use of STARTTLS among 116 Dutch organizations which found that: 55% of their domains used STARTTLS, 34% did not support STARTTLS, and 11% could not be tested. Our results provide another perspective, including how incoming messages are protected, mail is authenticated, and organizations deploy STARTTLS .

Mail Redirection There is a large corpus of work on DNS servers that provide false responses in order to facilitate content filtering [92, 167, 187, 245]. However, our study is the first to measure the extent to which DNS servers are falsifying MX records for mail providers and the amount of mail sent through these servers. In 2014, the Electronic Frontier Foundation (EFF), Golden Frog, and Telecom Asia posted anecdotal evidence of several ISPs blocking STARTTLS sessions in the United States and Thailand [115, 135, 236]. The EFF proposed STARTTLS Everywhere, an open source project that contains a public list of domains that support STARTTLS and scripts for generating configuration files for common MTAS that require STARTTLS for those domains [136]. Our work provides an additional perspective and estimates the amount of mail affected by STARTTLS stripping.

Internet-wide Scanning While Internet-wide scanning has not previously been used to measure the mail security ecosystem, it has become a standard practice for measuring the HTTPS ecosystem. In 2010, the EFF performed a distributed scan [102] of the IPv4 address space to identify certificate authorities. Later, in 2011, Holz et al. [137] scanned the Alexa

Top Million in order to measure HTTPS deployment and commonly used CAs. In 2012, Heninger et al. [133] performed comprehensive scans of HTTPS servers and detected wide use of weak cryptographic keys. Again in 2013, Durumeric et al. [99] completed daily scans in order to identify weaknesses in the HTTPS CA ecosystem. In 2014, Huang et al. [140] scanned the Top Million to measure the deployment of Forward Secrecy.

6.9 Conclusion

While electronic mail carries some of users' most sensitive correspondence, SMTP did not originally include support for message confidentiality or integrity. Over the past fifteen years, the mail community has retrofitted SMTP with several security mechanisms, including STARTTLS, SPF, DKIM, and DMARC. In this work, we analyzed the global adoption of these technologies using data from two perspectives: Internet-wide scans and logs of SMTP connections to and from one of the world's largest mail providers over a sixteen month period. Our measurements show that the use of these secure mail technologies has surged over the past year. However, much of this growth can be attributed to a handful of large providers, and many smaller organizations continue to lag in both deployment and proper configuration. The fail-open nature of STARTTLS and the lack of strict certificate validation reflect the need for interoperability amidst the gradual rollout of secure mail transport, and they embody the old adage that "the mail must go through." Unfortunately, they also expose users to the potential for man-in-the-middle attacks, which we find to be so widespread that they affect more than 20% of messages delivered to Gmail from several countries. We hope that by drawing attention to these attacks and shedding light on the real-world challenges facing secure mail, our findings will motivate and inform future research.

CHAPTER 7

Understanding Heartbleed's Impact

7.1 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. 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.

We also investigate widespread attempts to exploit Heartbleed, as seen in extensive bulk traffic traces recorded at four sites. We find no evidence of exploitation prior to the vulnerability's public disclosure, but we detect subsequent exploit attempts from almost 700 sources, beginning less than 24 hours after disclosure. Comparing attack attempts across sites, we observe that despite the large number of sources and scans, only a handful appear to reflect exhaustive Internet-wide scans.

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.

7.2 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.

7.2.1 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.

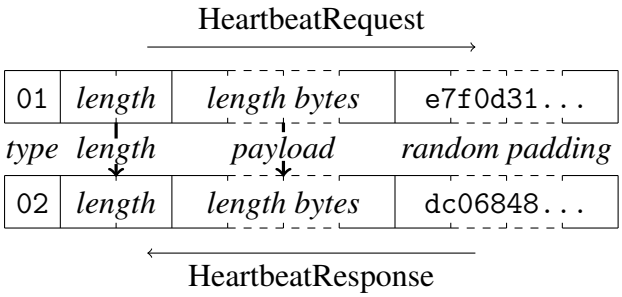


Figure 7.1: **Heartbeat Protocol.** Heartbeat requests include user data and random padding. The receiving peer responds by echoing back the data in the initial request along with its own padding.

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.

7.2.2 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 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 [225], 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 7.1.

7.2.3 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.

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 7.1: **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 [121, 196].

The OpenSSL patch adds a bounds check that discards the `HeartbeatRequest` message 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 [103, 204, 234].

7.2.4 Heartbleed Timeline

The Heartbleed vulnerability was originally found by Neel Mehta, a Google computer security employee, in March 2014 [121]. 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 [196], followed by the public security advisory [195] 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 [121]. 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 7.1.

7.3 The Impact of Heartbleed

Heartbleed had the potential to affect any service that used OpenSSL to facilitate TLS connections, including popular web, mail, messaging, and database servers (Table 7.2). To track its damage, we performed regular vulnerability scans against the Alexa Top 1 Million domains [1] and against 1% samples of the public, non-reserved IPv4 address space. We generated these samples using random selection with removal, per ZMap's existing randomization function [101]. We excluded hosts and networks that previously requested removal from our daily HTTPS scans [99]. In this section, we analyze the impact on those services—particularly HTTPS. We have publicly released all of the data used for this analysis at https://scans.io/study/umich-heartbleed.

7.3.1 Scanning Methodology

We tested for the Heartbleed bug by modifying ZMap [101] to send Heartbeat requests with no payload nor padding, and the length field set to zero. Per the RFC [225], these requests should be rejected. However, vulnerable versions of OpenSSL send a response containing only padding, rather than simply drop the request. The patched version of OpenSSL—as well as other popular libraries, including GnuTLS, NSS, Bouncy Castle, PolarSSL, CyaSSL and MatrixSSL—correctly discard the request (or do not support the Heartbeat Extension).

We emphasize that this approach does not exploit the vulnerability or access any private memory—only random padding is sent back by the server. While it was later found that Heartbleed scanning caused HP Integrated Lights-Out (iLO) devices to crash [14], we received no reports of our scans disrupting these devices—likely because our approach did not exploit the vulnerability. We have publicly released our scanner at https://zmap.io.

7.3.2 False Negatives

Our Heartbleed scanner contained a bug that caused vulnerable sites to sometimes appear safe due to a timeout when probing individual hosts. The root cause was that the scanner labelled each host's vulnerability as false by default, rather than null or unknown. If a Heartbleed test timed out, the scanner returned the host's vulnerability status as the default false, providing no indication of a failed test. The result is a potential false negative, where the scan reports the system as immune. Note that our scanner does not however err when reporting a system as vulnerable. As we develop in this section, the likelihood

of a given scan exhibiting such as false negative fortunately does not appear to depend on the particular address being scanned, and this allows us to estimate the false negative rate.

We first assessed whether some addresses were more prone to manifest false negatives than others. To do so, we compared three complete IPv4 scans and examined systems reported as vulnerable in one scan but immune in previous scans. Since the scanner does not err in reporting a host as vulnerable, any prior report of immunity reflects a false negative (assuming no one unpatches systems). We found the IP addresses associated with such false negatives spread evenly across the address space, without any apparent correlations. This observation leads us to believe the false negatives manifest in an address-independent manner.

Although our initial scanner did not fetch the web page itself, a subsequent change in the comprehensive scan (but not the incremental scans) attempted to fetch the server's home page. As the home page fetch occurred after the Heartbleed check, any reported home page data implicitly indicates that the Heartbleed test successfully completed without a timeout.

To investigate the false negative rate, we use two full scans of the IPv4 address space, one with and one without home page fetching. The full scan conducted on April 24 did not grab server home pages, while the May 1 scan did, hence we know the validity of scan results from the May 1 scan. To soundly conduct this comparison we need to remove servers that may have switched IP addresses between the two scans. To do so, we only considered servers that presented identical TLS certificates between the two scans. While this restriction potentially introduces a bias because some sites will have both patched and changed their TLS certificates, the address-independent nature of the false negatives should cause this effect to even out.

Our scanner failed to grab the server home page for 24% of the hosts in the May scan. Of these 24% of hosts, we observe 44% appear immune. False negatives could only have occurred when testing these hosts. The remaining 56% of hosts appeared vulnerable (and hence are correctly labelled). From this, we conclude that at most $(0.24 \cdot 0.44) = 0.105$, or 10.5%, of hosts were incorrectly labelled in the May 1 scan.

For the April scan, the only observable signal of a false negative is if a host was reported immune and then reported vulnerable in the May scan. We find 6.5% of hosts exhibit this behavior. Assuming that people do not unpatch their systems, this provides an estimated lower bound of 6.5% for the April scan false negative rate. This estimate represents a lower bound because we cannot determine the vulnerability in April of a host that appears immune in both scans. In that case, a false negative is a host vulnerable in April but reported as immune, and patched by May. However, we do ob-

serve that of hosts reported as vulnerable in the April scan and successfully tested in May (so the server page was retrieved), only 0.36% appeared immune in May, indicating a very modest overall patching rate between the two scans (which accords with Figure 7.3 below). Given that our false negatives are address-independent, we expect a similarly low patch rate for all vulnerable April hosts. Thus, while a 6.5% false negative rate is a lower bound for the April scan, the true rate should not be significantly higher.

Given the similarity of these two false negative estimates using two different scans, ultimately we conclude that the scanner exhibited a false negative rate between 6.5% and 10.5%, but that these manifest independently of the particular server scanned. Due to this address-independent behavior, we can assume a similar false negative rate for sampled scans. We attempt to account for this error whenever possible. In particular, the bias implies that any population-based survey based on a single scan underestimates the vulnerable population.

Web Servers		Mail Servers		Database Servers		XMPP Servers		Other Servers	
Apache (mod_ssl) [177]	Yes	Sendmail [210]	Yes	MySQL [210]	Yes	OpenFire [15]	No	OpenVPN [197]	Yes
Microsoft IIS [178]	No	Postfix [210]	Yes	PostgreSQL [210]	Yes	Ejabberd [6]	Yes	OpenLDAP [212]	Yes
Nginx [17]	Yes	Qmail [210]	Yes	SQL Server [178]	No	Jabberd14 [250]	Yes	Stunnel [222]	Yes
Lighttpd [210]	Yes	Exim [120]	Yes	Oracle [198]	No	Jabberd2 [145]	Yes	Openswan [191]	Yes
Tomcat [20]	Yes	Courier [126]	Yes	IBM DB2 [141]	No			Telnetd-ssl [5]	Yes
Google GWS [193]	Yes	Exchange [178]	No	MongoDB [184]	Yes			OpenDKIM [3]	Yes
LiteSpeed [164]	Yes	Dovecot [120]	Yes	CouchDB [110]	No			Proftpd [221]	Yes
IBM Web Server [141]	Yes	Cyrus [186]	Yes	Cassandra [7]	No			Bitcoin Client [53]	Yes
Tengine [16]	Yes	Zimbra [199]	Yes	Redis [120]	No				
Jetty [194]	No								

Table 7.2: **Vulnerable Server Products.** We survey which server products were affected by Heartbleed.

7.3.3 Impact on Popular Websites

Determining which websites were initially vulnerable poses significant difficulties. Little attention was paid to the Heartbeat Extension prior to the vulnerability announcement, and many popular sites patched the vulnerability within hours of the disclosure. Codenomicon, one of the groups that discovered Heartbleed, speculated that 66% of HTTPS sites were vulnerable [177]. However, this number represented the Apache and Nginx market share and may well reflect an overestimate, because some operators may have disabled the extension, deployed dedicated SSL endpoints, or used older, non-vulnerable versions of OpenSSL.

7.3.3.1 Top 100 Websites

All of the Alexa Top 100 websites were patched within 48 hours of disclosure—prior to the start of our scans. To document the impact on these websites, we aggregated press releases,

other's targeted scans, and quotes provided to Mashable, a news site that hosted one of the popular lists of sites for which users should change their passwords due to possible exposure via Heartbleed [13].

Al-Bassam completed a vulnerability scan of the Alexa Top 10,000 domains on April 8, 2014 at 16:00 UTC (22 hours after the vulnerability disclosure) [29]. His scan found 630 vulnerable sites, 3,687 supporting HTTPS but not vulnerable, and 5,683 not supporting HTTPS. Several prominent sites, including Yahoo, Imgur, Stack Overflow, Flickr, Sogou, OkCupid, and DuckDuckGo, were found vulnerable. We investigated other sites in the Alexa Top 100 and found that half made a public statement regarding vulnerability or provided information to Mashable [11, 13, 21, 22, 29, 31, 37, 80, 91, 113, 142, 143, 193, 202, 209, 240, 241, 252, 258, 259]. Combining these press releases, Mashable's report, and Al-Bassam's scan, we find that at least 44 of the Alexa Top 100 websites were vulnerable. However, this figure reflects a lower bound, as we were unable to find information for some sites. Table 7.3 lists the vulnerability status of the top 30 HTTPS-enabled sites in the US.

Site	Vuln.	Site	Vuln.	Site	Vuln.
Google	Yes	Bing	No	Wordpress	Yes
Facebook	No	Pinterest	Yes	Huff. Post	?
Youtube	Yes	Blogspot	Yes	ESPN	?
Yahoo	Yes	Go.com	?	Reddit	Yes
Amazon	No	Live	No	Netflix	Yes
Wikipedia	Yes	CNN	?	MSN.com	No
LinkedIn	No	Instagram	Yes	Weather.com	?
eBay	No	Paypal	No	IMDB	No
Twitter	No	Tumblr	Yes	Apple	No
Craigslist	?	Imgur	Yes	Yelp	?

Table 7.3: **Vulnerability of Top 30 US HTTPS-Enabled Websites.** We aggregate published lists of vulnerable sites, press releases, and news sources to determine which of the top sites were vulnerable before the discovery of Heartbleed.

7.3.3.2 Estimating Broader Impact

Within 48 hours of the initial disclosure, we conducted our first vulnerability scan of the Alexa Top 1 Million. At that point, we found that 45% of all sites supported HTTPS. 60% of those supported the Heartbeat Extension, and 11% of all HTTPS sites were vulnerable. While 60% of HTTPS sites supported the extension, 91% of these were powered by known vulnerable web servers (e.g., Nginx or Apache Web Server), as shown in Table 7.4. If all of these servers were initially vulnerable and operators installed a patched OpenSSL version (rather than rebuilding OpenSSL with Heartbeat disabled), at most about 55% of the HTTPS sites in the Alexa Top 1 Million were initially vulnerable.

While disabling the largely unused extension would appear to provide an obvious solution, it is not possible to disable the extension through a configuration file. Instead, this change requires recompiling OpenSSL with a specific flag—an option likely more laborious than updating the OpenSSL software package.

Some sites may possibly have used an older version of OpenSSL that was not vulnerable. To estimate a lower bound for the number of vulnerable sites, we considered sites that used vulnerable web servers and supported TLS 1.1 and 1.2—features first introduced in OpenSSL 1.0.1 along with the Heartbeat Extension. Such sites would have been vulnerable unless administrators had recompiled OpenSSL to explicitly disable the extension.

To estimate the number of sites that supported TLS 1.1 and 1.2 prior to the Heartbleed disclosure, we analyzed the data collected by the Trustworthy Internet Movement's SSL Pulse [19], which provides monthly statistics of SSL-enabled websites within the Alexa Top 1 Million. We find that 56,019 of the 171,608 (32.6%) sites in the SSL Pulse dataset supported TLS 1.1 or 1.2. Of these sites, 72.7% used known vulnerable web servers, yielding an estimated lower bound of 23.7% of the sites being vulnerable.

In summary, we can reasonably bound the proportion of vulnerable Alexa Top 1 Million HTTPS-enabled websites as lying between 24–55% at the time of the Heartbleed disclosure.

Web Server	Alexa Sites	Heartbeat Ext.	Vulnerable
Apache	451,270 (47.3%)	95,217 (58.4%)	28,548 (64.4%)
Nginx	182,379 (19.1%)	46,450 (28.5%)	11,185 (25.2%)
Microsoft IIS	96,259 (10.1%)	637 (0.4%)	195 (0.4%)
Litespeed	17,597 (1.8%)	6,838 (4.2%)	1,601 (3.6%)
Other	76,817 (8.1%)	5,383 (3.3%)	962 (2.2%)
Unknown	129,006 (13.5%)	8,545 (5.2%)	1,833 (4.1%)

Table 7.4: **Alexa Top 1 Million Web Servers.** We classify the web servers used by the Alexa Top 1 Million Sites, as observed in our first scan on April 9, 2014. Percentages represent the breakdown of server products for each category. We note that while Microsoft IIS was not vulnerable to Heartbleed, a small number of IIS servers used vulnerable SSL terminators.

7.3.4 Pre-Disclosure Patching

Google, Akamai, and other sites disabled the Heartbeat Extension prior to public disclosure. To detect when services disabled the Heartbeat Extension, we examined data from the ICSI Certificate Notary, which passively monitors TLS connections from seven research and university networks (approximately 314K active users) [32].

The Notary data shows that Google disabled Heartbeat starting at least 12 days prior to public disclosure, with all servers Heartbeat-disabled by April 15. While some servers

still had Heartbeat enabled after disclosure, they may not have been exploitable. Google may have already patched those servers, and decided afterwards to disable the Heartbeat Extension as a company-wide policy. Similarly, Akamai began disabling Heartbeat at least 4 days prior to disclosure, completing the process by April 18.

7.3.5 Internet-Wide HTTPS Vulnerability

We began performing daily 1% scans of the IPv4 address space on April 9, 48 hours after the disclosure. Our first scan found that 11.4% of HTTPS hosts supported the Heartbeat Extension and 5.9% of all HTTPS hosts were vulnerable. Combining these proportions from random sampling with our daily scans of the HTTPS ecosystem [99] (which do not include Heartbleed vulnerability testing), we estimate that 2.0 million HTTPS hosts were vulnerable two days after disclosure.

Surprisingly, 10 ASes accounted for over 50% of vulnerable HTTPS hosts but represented only 8.6% of all HTTPS hosts (Figure 7.2). With the exception of Comcast Cable Communications, the ASes all belonged to web hosting companies or cloud providers (Table 7.5). The vulnerable hosts in the Comcast AS were Fortinet devices. In the case of Strato Hosting, vulnerable addresses were hosting Parallels Plesk Panel, a web hosting management software. The vulnerable addresses of Minotavar Computers, ZeXoTeK IT-Services, Euclid systems, Vivid Hosting, and ACCESSPEOPLE-DE all served the default Apache page, likely reflecting named-based virtual hosts. In the case of the two Amazon ASes and Hetzner Online, a large number of the vulnerable hosts served public facing websites, and used Apache or Nginx.

AS	% of Vulnerable	% of HTTPS
Minotavar Computers EOOD	18.5%	1.7%
ZeXoTeK IT-Services GmbH	13.0%	0.9%
ACCESSPEOPLE-DE ISP-Service	7.4%	0.7%
Amazon.com, Inc.	4.6%	0.8%
Amazon.com, Inc.	4.1%	0.9%
Hetzner Online AG	2.6%	0.4%
Comcast Cable Communications	2.3%	2.8%
Vivid Hosting	2.0%	0.1%
Euclid Systems	1.5%	0.1%
Strato Hosting	1.4%	0.1%
Total	57.4%	8.6%

Table 7.5: **Top ASes with Most Vulnerable Hosts.** We aggregate hosts by AS and find that 57% of vulnerable hosts in the April 9 scan were located in only 10 ASes.

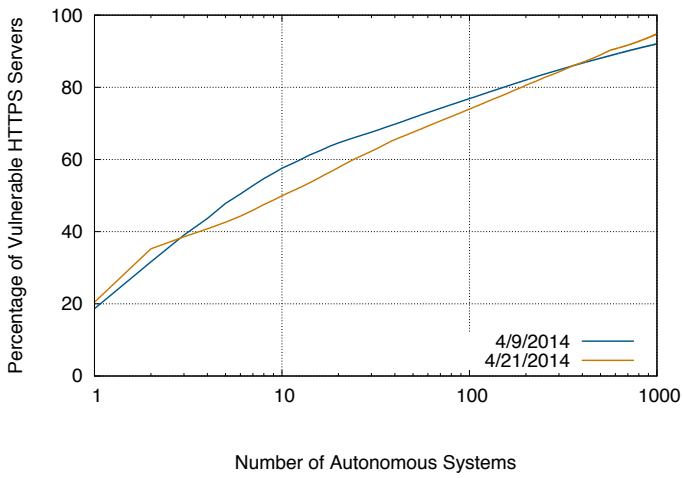


Figure 7.2: **Vulnerable Servers by AS.** We aggregate vulnerable hosts by AS and find that over 50% of vulnerable hosts are located in ten ASes.

7.3.6 Vulnerable Devices and Products

Heartbleed also affected many embedded systems, including printers, firewalls, VPN endpoints, NAS devices, video conferencing systems, and security cameras. To understand the embedded systems that were affected, we analyzed the self-signed certificates employed by vulnerable hosts. We clustered these by fields in the certificate Subject and manually inspected large clusters. From this, we developed device "fingerprints". We took a conservative approach and chose the most restrictive fingerprints in order to minimize false positive identifications. This, and the manual effort required, means that our fingerprints lack comprehensive coverage. However, we still identified 74 distinct sets of vulnerable devices and software packages that fall into a number of broad categories:

Communication Servers: IceWarp messaging, Zimbra collaboration servers, iPECS VoIP systems, and Polycom and Cisco video conference products.

Software Control Panels: Puppet Enterprise Dashboard, IBM System X Integrated Management Modules, Kloxo Web hosting control panel, PowerMTA, Chef/Opscode management consoles, VMWare servers, and Parallels control panels for Plesk and Confixx.

Network Attached Storage: QNAP, D-Link, ReadyNAS, LaCie, Synology, and Western Digital NAS devices.

Firewall and VPN Devices: Devices from Barracuda Networks, Cisco, SonicWALL, WatchGuard, OpenVPN, pfSense, TOPSEC Network Security (a Chinese vendor), and Fortinet.

Printers: Dell, Lexmark, Brother, and HP printers.

Miscellaneous: Hikvision and SWANN security cameras, AcquiSuite power monitors, IPACCT (a management tool used by Russian ISPs), Aruba Networks WiFi login portals, INSYS VPN access for industrial controllers, and SpeedLine Solutions (the "#1-rated Pizza POS System").

7.3.7 Other Impacts

While our study focuses on Heartbleed's impact on public HTTPS web services, Heartbleed also affected mail servers, the Tor network, Bitcoin, Android, and wireless networks, as we briefly assess in this section.

Mail Servers. SMTP, IMAP, and POP3 can use TLS for transport security via use of a `StartTLS` directive within a plaintext session. As such, if mail servers used OpenSSL to facilitate TLS connections, they could have been similarly vulnerable to Heartbleed. On April 10, we scanned a random 1% sample of the IPv4 address space for vulnerable SMTP servers. We found that 45% of those providing SMTP+TLS supported the Heartbeat Extension, and 7.5% were vulnerable to Heartbleed.

These estimates only provide a lower bound, because similar to HTTPS, our scanner sometimes timed out, causing false negatives. (We also scanned for IMAP and POP3 servers, but later analysis of the data found systematic flaws that rendered the results unusable.)

Tor Project. Tor relays and bridges use OpenSSL to provide TLS-enabled inter-relay communication. In our April 10 scan, we found that 97% of relays supported Heartbeat and could have been vulnerable. 48% of the relays remained vulnerable at that time, three days after announcement of the vulnerability. The vulnerability could have allowed an attacker to extract both short-term onion and long-term identity keys, ultimately allowing an attacker to intercept traffic and impersonate a relay. In the case of a hidden service, the bug could have allowed an entry guard to locate a hidden service. The Tor client was similarly affected, potentially allowing entry guards to read sensitive information from a client's memory, such as recently visited websites [89].

Bitcoin Clients. Heartbleed affected both Bitcoin clients and exchanges, and in the most severe case, allowed an attacker to compromise the accounts on a popular exchange, BTCJam [18]. The value of Bitcoin did not change drastically on April 7, the date of public disclosure, falling only 3.1% from the previous day (a figure within its regular volatility) and returned to its April 6 value by April 14.

All versions of the Bitcoin software from May 2012 to April 2014 used a vulnerable OpenSSL version [2]. Immediately after Heartbleed's disclosure, a new Bitcoin version was released linking to the newly patched OpenSSL version. Because clients were also

affected by the bug, attackers could potentially compromise wallets or retrieve private keys if a susceptible user followed a payment request link [53]. However, we have not found any reports of the theft of Bitcoins from local wallets.

Several companies, including Bitstamp and Bitfinex, temporarily suspended transactions on April 8 until they could patch their servers. In the most severe case, 12 customers had a total of 28 BTC (≈ $6,500) stolen from BTCJam after account credentials were compromised, though with all funds subsequently reinstated by the exchange [18].

Android. Heartbleed only affected Android version 4.1.1 [193]. It is unclear how many devices currently run the affected version, but Google recently estimated that 33.5% of all Android devices run Android 4.1.x in April 2014 [10]. A vulnerable device would have been susceptible to having memory read by a malicious server.

Wireless Networks. Several variants of the Extended Authentication Protocol, a commonly used framework for wireless network authentication, use TLS, including EAP-PEAP, EAP-TLS, and EAP-TTLS. For implementations based on OpenSSL, Heartbleed would have allowed attackers to retrieve network keys and user credentials from wireless clients and access points [118]. We do not know of any statistics regarding what sort of vulnerable population this potentially represents.

7.4 Patching

In Section 7.3, we estimated the initial impact of Heartbleed. In this section, we discuss the patching behavior that occurred subsequent to the disclosure.

7.4.1 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. As discussed in Section 7.3, 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 7.3, 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

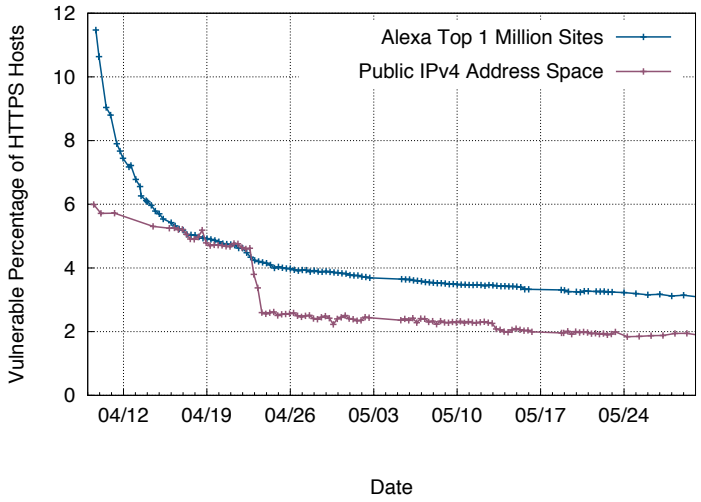


Figure 7.3: **HTTPS Patch Rate.** We track vulnerable web servers in the Alexa Top 1 Million and the public IPv4 address space. We track the latter by scanning independent 1% samples of the public IPv4 address space every 8 hours. Between April 9 and June 4, the vulnerable population of the Alexa Top 1 Million shrank from 11.5% to 3.1%, and for all HTTPS hosts from 6.0% to 1.9%.

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 7.3.

7.4.2 Internet-Wide HTTPS

As can be seen in Figure 7.3, 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.

7.4.3 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

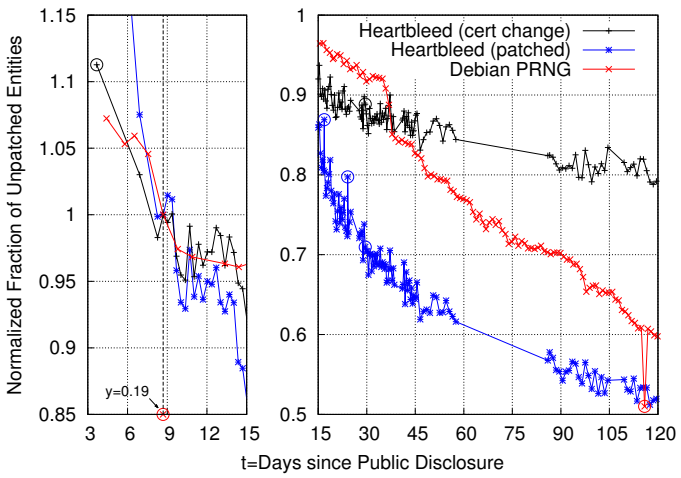


Figure 7.4: **Debian PRNG vs. Heartbleed Patch Rate.** The y-axis is normalized at 8.7 days, indicated by the vertical striped line. Thus, the fraction of unpatched entities at a given time is relative to the fraction at 8.7 days after disclosure, for each dataset. Except for the points marked by ○, 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.

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 [255]. Given the similarities in the severity and nature of remediation between this event and Heartbleed, we compared the community's responses to both disclosures.

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 [255], in which groups were tracked as a whole from the first measurement.

Figure 7.4 shows for both datasets the fraction of unfixed entities to the total number of 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.[1] 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 7.4. 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.

7.5 Certificate Ecosystem

Heartbleed allowed attackers to extract private cryptographic keys [234]. As such, the security community recommended that administrators generate new cryptographic keys and revoke compromised certificates [120]. In this section, we analyze to what degree operators followed these recommendations.

7.5.1 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 [99], and ICSI's

[1] See Section 7.5.1 for a discussion on the replacement of public/private key pairs in addition to certificates.

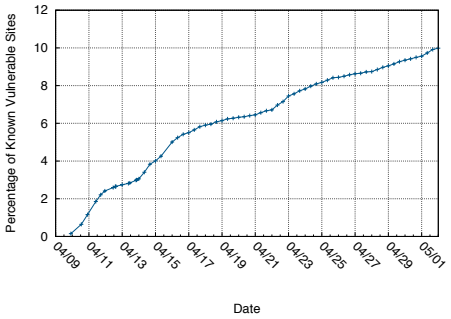


Figure 7.5: **Certificate Replacement on Vulnerable Alexa Sites.** We monitored certificate replacement on vulnerable Alexa Top 1 Million sites and observe only 10% replaced certificates in the month following public disclosure.

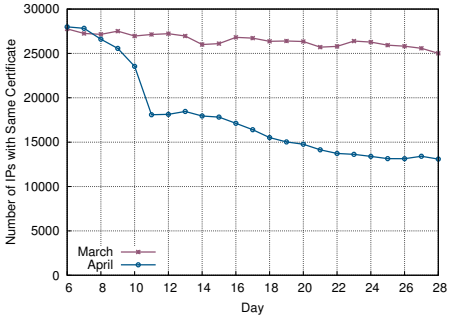


Figure 7.6: **ICSI Notary Certificate Changes.** Over both March and April, we track the number of servers who have the same certificate as on the 6th of each month. We only include servers that served the same certificate for the previous month.

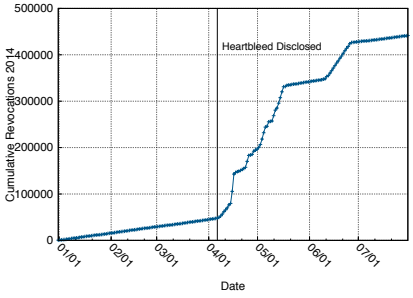


Figure 7.7: **Revocations after Heartbleed.** Certificate revocations dramatically increased after the disclosure. The jumps reflect GlobalSign (first) and GoDaddy (rest). However, still only 4% of HTTPS sites in the Alexa Top 1 Million revoked their certificates between April 9 and April 30, 2014.

Certificate Notary service [32]. 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 7.5). For comparison, we observed that 73% of vulnerable hosts detected on April 9 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 (see Section 7.6.1) 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 7.6, 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.

7.5.2 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.

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 7.7, 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 [207].

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's revocations. 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 [207]. StartCom, a CA that offers free SSL certificates, came under fire for continuing their policy of charging for revocation after the Heartbleed disclosure [172]. However, revocation places a sizable financial strain on CAs due to bandwidth costs [12, 207].

7.5.3 Forward Secrecy

Heartbleed highlights the importance of using *forward secret* cipher suites in TLS. Without forward secrecy, the compromise of a server's private key, such as due to Heartbleed, allows an attacker who recorded previous communications encrypted with TLS to recover the session key used to protect that communication, subverting its confidentiality. Thus, we might expect operators to respond to Heartbleed by ensuring that at least in the future their servers will support forward secrecy.

Unfortunately, we find that only 44% of the connections observed by the ICSI Notary in May 2014 used forward secrecy. There has been a slow increase in adoption between December 2013 and April 2014, a gain of 1.0–4.3% each month. We observe a 4.2% increase between March and April 2014, but no larger than that between January and February. Surprisingly, this trend stagnated from April to August; the percentage of connections using forward secrecy stayed virtually the same. Currently, we are not sure why the increase in forward secrecy cipher use ceased. However, it appears clear that Heartbleed did not spur adoption of forward secrecy.

7.6 Attack Scene

In addition to tracking vulnerable servers, we analyzed who was scanning for the Heartbleed vulnerability by examining network traffic collected from passive taps at Lawrence Berkeley National Laboratory (LBNL), the International Computer Science Institute (ICSI), and the National Energy Research Scientific Computing Center (NERSC), as well as a honeypot operated by a colleague on Amazon EC2.

To detect Heartbleed scanning, we extended the Bro's SSL/TLS analyzer to recognize Heartbeat messages [61, 201]. Note that this approach parses the full TLS protocol data stream, including the TLS record layer which remains unencrypted throughout the session, and thus achieves an accuracy significantly better than that provided by simple byte pattern matching. We have released our Bro modifications along with our detection script via the Bro git repository.

7.6.1 Pre-Disclosure Activity

LBNL's network spans two /16s, one /20 and one /21. The institute frequently retains extensive packet traces for forensic purposes, and for our purposes had full traces available from February–March 2012, February–March 2013, and January 23–April 30 2014. ICSI uses a /23 network, for which we had access to 30-days of full traces from April 2014. NERSC has a /16 network, for which we analyzed traces from February to April 2014. The EC2 honeypot provided full packet traces starting in November 2013.

For all four networks, over these time periods our detector found no evidence of any exploit attempt up through April 7, 2014. This provides strong evidence that at least for those time periods, no attacker with prior knowledge of Heartbleed conducted widespread scanning looking for vulnerable servers. Such scanning however could have occurred during other time periods.

7.6.2 Post-disclosure Activity

To detect post-disclosure scanning, we similarly examined packet traces from LBNL, ICSI, and the EC2 honeypot. The first activity we observed originated from a host at the University of Latvia on April 8, starting at 15:18 UTC (21 hours 29 minutes after public disclosure), targeting 13 hosts at LBNL. This first attack was unusual in that it sent both unencrypted (pre-handshake) and encrypted (post-handshake) Heartbleed exploit packets to each host, likely trying to gauge the effectiveness of both approaches. We observed scanning of the other two networks within a few more hours.

In total, we observed 5,948 attempts to exploit the vulnerability from 692 distinct hosts (Table 7.7). These connections targeted a total 217 hosts. 7 attackers successfully completed 103 exploit attempts against 12 distinct hosts (excluding the intentionally vulnerable honeypot). Figure 7.8 presents the temporal behavior of scanning as seen at each location.

We detected several different types of exploit attempts, which we list in Table 7.6. In types (1) and (2), attackers sent exploit attempts prior to completely establishing the TLS session, which allowed us to directly inspect the attack payload. After establishment of the TLS session, the only pieces of information we can retrieve are the message

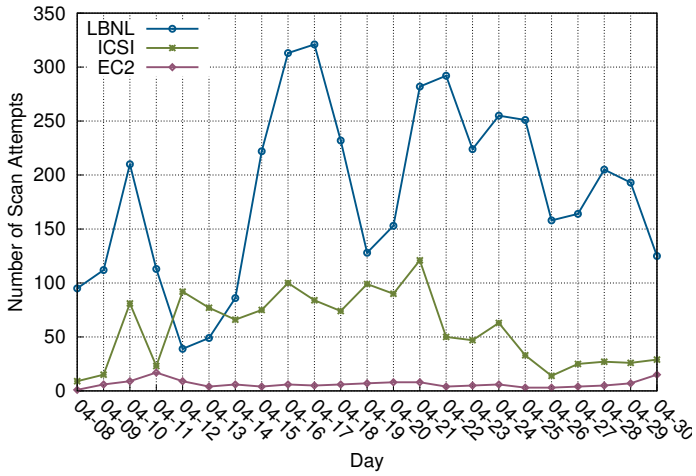


Figure 7.8: **Incoming Attacks.** We track the number of incoming connections containing Heartbleed exploit messages seen at ICSI, LBNL, and the EC2 honeypot per day.

Type	Sources	Targets	Connections
(1) Cleartext attacks	628	191	1,691
(2) Successful cleartext attacks	4	10	26
(3) Short Heartbeats	8	10	187
(4) Heartbeats before data	61	132	4,070
(5) Successful attacks after encryption	5	6	77
Total	692	217	5,948

Table 7.6: **Attack Types.** We list different stages of attacks observed against LBNL, ICSI and the EC2 honeypot.

type and size, which are both transferred in the clear. To detect scanning conducted in a fashion similar to our own, we checked the length of the encrypted message (3). We also consider all Heartbeat messages that we see prior to the first transmission of application data (4) to reflect exploit attempts.[2] We identify attacks as successful when the destination server responds with more data than included in the original request (5), though as noted above we do not consider attacks on the EC2 honeypot as "successful".

The sixteen most aggressive probe sources (by number of scans) reside in Amazon address space. The scans originating from these hosts share many characteristics, including the use of encryption, the same packet length, and identical cipher suites. Closer examination reveals that these hosts belong to two popular Heartbleed scan services: `filippio.io` (3,964 attempts from 40 hosts) and `ssllabs.com` (16 attempts from 5 hosts).

[2]Heartbeat messages should never precede application data. We verified this does not happen in real-world traffic by manually reviewing traces prior to the Heartbleed disclosure. We observed no such instances.

Examining connection attempts across each site, we find only three sources that attempted to scan all three networks. However, LBNL aggressively blocks scan traffic, so most hosts scanning it were likely blocked before they could locate a server against which to attempt Heartbleed exploits. Considering only the EC2 honeypot and the ICSI network (neither of which block scanning), we find 11 sources that scanned both.

Apart from our scans at the University of Michigan and another research group at TU Berlin, we found four sources that targeted more than 100 addresses at ICSI. Two of the sources were located in CHINANET, one at Nagravision, and one at Rackspace. The remaining Heartbleed exploit attempts only targeted smaller numbers of hosts without performing widespread scanning.

Hosts performing widespread scans exclusively targeted port 443 (HTTPS). We observed a small number of exploit against ports 993 (IMAPS), 995 (POP3S), 465 (SMTPS), as well as GridFTP. We also found a small number of exploit attempts against services running on other ports.

While we observed Heartbleed attacks originating from a large number of sources, we find that most hosts did not target more than one of our sites and likely do not represent Internet-wide scanning. Given the low volume of widespread scanning, the 201 sources attempting to exploit the EC2 honeypot appears surprisingly high. Our hypothesis is that attackers may preferentially scan denser address spaces, such as those of Amazon EC2, as they will likely yield a greater number of vulnerable targets.

7.7 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 7.5, 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

AS Name	ASN	Scans	Hosts
Amazon.com	14618	4,267	206
China Telecom	4812	507	139
China169 Backbone	4837	147	34
Chinanet	4134	115	23
University of Michigan	36375	92	3
SoftLayer	36351	85	2
University of Latvia	8605	50	1
Rackspace	19994	47	11
GoDaddy.com	26496	34	15
OVH	16276	30	9
ViaWest	13649	30	1
Guangdong Mobile	9808	29	1
New York Internet Company	11403	29	1
ServerStack	46652	27	1
Comcast	7922	20	5
Global Village Telecom	18881	19	4
China Telecom	4816	16	1
Turk Telekomunikasyon	9121	15	8
DFN	680	15	2
Amazon.com	16509	12	5
TekSavvy Solutions	5645	11	2
Oversun	48172	11	2
HKNet	4645	11	1
CariNet	10439	10	8
PowerTech	5381	10	1
Nagravision	42570	10	1
CYBERDYNE	37560	10	1

Table 7.7: **ASes Responsible for Attacks.** We list the ASNs/ISPs that sent 10 or more Heartbleed exploit attempts to LBNL, ICSI, and our EC2 honeypot.

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 [207]. 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 [171]. Our community needs to determine effective support models for these core open-source projects.

Vulnerability Disclosure. With the exception of a small handful, the most prominent websites patched within 24 hours. In many ways, this represents an impressive feat. However, we also observed vulnerability scans from potential attackers within 22 hours, and it is likely that popular sites were targeted before the onset of large, indiscriminate scans.

Several factors indicate that patching was delayed because the Heartbleed disclosure process unfolded in a hasty and poorly coordinated fashion. Several major operating system vendors were not notified in advance of public disclosure, ultimately leading to

delayed user recovery. As discussed in Section 7.3, a number of important sites remained vulnerable more than 24 hours after initial disclosure, including Yahoo, the fourth most popular site on the Internet. The security community needs to be better prepared for mass vulnerability disclosure before a similar incident happens again. This includes addressing difficult questions, such as how to determine which software maintainers and users to notify, and how to balance advance disclosure against the risk of premature leaks.

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.

7.8 Conclusion

In this work we analyzed numerous aspects of the recent OpenSSL Heartbleed vulnerability, including (1) who was initially vulnerable, (2) 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 investigated the attack landscape, finding no evidence of large-scale attacks prior to the public disclosure, but vulnerability scans began within 22 hours. We observed post-disclosure attackers employing several distinct types of attacks from 692 sources, many coming from Amazon EC2 and Chinese ASes. 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 8

Conclusion and Future Work

As the Internet plays an increasingly critical role in our society, understanding the dynamics of and securing the plethora of devices and services that constitute it has become of paramount importance. While research techniques like passive observation and random sampling enabled researchers to understand many aspects of the Internet's day-to-day operation, these approaches have typically focused on the most popular services or on a small demographic of users, not on providing a comprehensive view. Beyond understanding the dynamics of individual devices, recent research has shown that certain security phenomena can only be easily observed using a near-global perspective [133]. This dissertation shows how fast Internet-wide scanning helps provide this missing perspective and enables researchers to answer a variety of new security questions.

I first showed that it is possible to efficiently scan the full public IPv4 address space by introducing ZMap—a network scanner specifically architected for large-scale research studies. ZMap is capable of surveying the entire IPv4 address space from a single machine in under an hour. Compared to Nmap—an excellent general-purpose network scanner and an industry standard that has been used in previous studies—ZMap achieves much higher performance and coverage, both completing scans more than 1300 times faster than Nmap's most aggressive settings and with 15% higher coverage of listening hosts. It achieves this by operating asynchronously with minimal per-target state, overloading network protocol fields to validate probe responses, and using a custom user-space network stack. I experimentally showed that ZMap is capable of scanning the public IPv4 address space on a single port in under 45 minutes, at 97% of the theoretical maximum speed for gigabit Ethernet and with an estimated 98% coverage of publicly available hosts.

While ZMap drastically reduced the time required to conduct large-scale scans, collecting meaningful data through Internet-wide scanning remained a specialized and labor-intensive process, which reduced the methodology's utility. To enable researchers to efficiently ask questions about the composition of the Internet, I architected Censys: a cloud-based service that maintains an up-to-date snapshot of the hosts and services on the public IPv4

address space. Then I showed how Censys can be used to quickly answer common security questions.

Second, I showed how the perspective provided by Internet-wide scanning can help: identify classes of weaknesses that only emerge at scale, uncover unexpected attacks, shed light on previously opaque distributed systems, and understand the impact of consequential vulnerabilities. I specifically highlighted four case studies that build on Internet-wide scanning:

1. **Uncovering Widespread Weak Keys in Network Devices** Randomness is essential for modern cryptography, and systems can fail catastrophically when used with a malfunctioning random number generator (RNG). In this first case study, I conducted the first Internet-wide survey to detect cryptographic keys compromised by weak RNGs, and found that they are surprisingly widespread. I investigated the devices producing weak keys, uncovering vulnerabilities in the Linux random number generator that would have been nearly impossible to find through reverse engineering or by analyzing a small sample of keys.

2. **Identifying Attacks on Email Delivery** As originally conceived, SMTP—the network protocol responsible for relaying messages between mail servers—did not authenticate senders or encrypt mail in transit. Instead, these features were later introduced as protocol extensions. Under the assumption that man-in-the-middle attackers do not exist between large Internet service providers, operators did not prioritize their deployment and standards bodies did not harden them to protect against active adversaries. In this case study, I showed how anolomies found during regular Internet-wide scans provided the evidence needed to uncover attacks in large ISPs that both block the encryption of mail and redirect messages for later inspection.

3. **Analyzing the HTTPS Certificate Ecosystem** HTTPS is based on the TLS encrypted transport protocol and a supporting PKI composed of thousands of certificate authorities (CAs)—entities that are trusted by users' browsers to vouch for the identity of web servers. Despite their critical role, there is no published list of these authorities. In the third case study, I showed how by collecting certificates through Internet-wide scanning, it is possible to formally identify the organizations that can sign certificates, identify their worrisome practices, and quantify current risks in the PKI.

4. **Understanding Vulnerability Impact** The Heartbleed vulnerability took the Internet by surprise in April 2014. The bug—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. In this last case study, I showed how fast Internet-wide scanning can help researchers immediately respond to severe vulnerabilities, including tracking the vulnerable population and monitoring patching behavior.

While ZMap drastically reduced the time needed to complete scans of the IPv4 address space, exhaustive scanning is becoming less effective as IPv4 addresses are depleted [242] and there remain a number of open research questions related to scanning.

IPv6 Scanning ZMap leverages the fact that increased network speeds and computational power enables comprehensively scanning *all* possible IPv4 addresses in a reasonable time period. However, this brute-force approach does not scale to IPv6, which is nearly 2^{96} times larger. As IPv6 deployment continues to increase, we will need to develop better approaches for inferring how operators assign IPv6 addresses, algorithms to predict addresses that should be scanned in the future, and models to understand the type of coverage we can achieve in the IPv6 space.

Name-based Scanning IPv4 exhaustion has not only led to increased IPv6 deployment, but also to multiplexing multiple sites and services on a single IP address. For example, TLS has introduced the Server Name Indication (SNI) extension, which allows HTTPS clients to specify the destination domain during the TLS handshake [55]. In turn, operators can host multiple HTTPS hosts on a single IP address. Recent work has shown that much of the HTTPS ecosystem is no longer accessible without name indication: only 30–40% of known browser-trusted X.509 certificates can be downloaded using IP scanning without SNI [242]. The expected global deployment of TLS 1.3—which does not permit connections without a specified name—will further exacerbate this problem. We will need to transition to name-based scanning, curating collections of known names, predicting and discovering new names, and understanding the biases introduced with no longer being able to comprehensively scan the entire search space.

Identifying Heterogeneous Devices As recently demonstrated by analyses of the Mirai botnet [34] and Internet-connected industrial control systems [180], embedded devices now play a crucial role in Internet security. The increasing heterogeneity of devices on the public Internet increases several challenges. First, we lack approaches to externally identify devices without expensive, manual labeling. We need to develop algorithms for automatically classifying devices if we are to reason about this emerging space. Second, devices are employing a wide-range of manufacturer- and industry-specific protocols. Today, scanning for these protocols requires reverse engineering proprietary protocols and manually

re-implementing them—both expensive and time consuming processes. In the future, we need to develop approaches for quickly deploying scanners to analyze these devices.

Exposing Meaningful Data While Censys exposes raw Internet-wide scan data to security researchers, many users are interested in using scan data to secure their own networks. We need to understand what data administrators need to best secure their networks, how to present this data in a constructive format, and how to best notify administrators of vulnerabilities.

Automated Approaches Despite advances in data collection techniques like ZMap and Certificate Transparency [158], security vulnerabilities and abuses within the PKI continue to be found years later during unrelated research. We need to develop automated approaches for uncovering these problems in real time instead of relying on manual, ad-hoc investigation.

The last few years have represented a unique period in the history of the Internet. Networks were fast enough to quickly scan the entire public IPv4 address space while the contention over IPv4 addresses was low enough that services could be accessed by IP. Fast Internet-wide scanning has enabled the research community to tackle a wide range of security problems, from uncovering attacks against email delivery to understanding how countries censor websites to identifying abuses within the HTTPS PKI. Unfortunately, this period is quickly coming to an end, if it hasn't already, and the research community needs to develop new approaches for name-based and IPv6 scanning in order to continue performing these empirical studies.

BIBLIOGRAPHY

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

[2] Bitcoin Core Version History. https://bitcoin.org/en/version-history.

[3] Installing OpenDKIM. http://www.opendkim.org/INSTALL.

[4] random(4) Linux manual page. http://www.kernel.org/doc/man-pages/online/pages/man4/random.4.html.

[5] Telnet Server with SSL Encryption Support. https://packages.debian.org/stable/net/telnetd-ssl.

[6] Install Ejabberd, Oct. 2004. http://www.ejabberd.im/tuto-install-ejabberd.

[7] Cassandra Wiki - Internode Encryption, Nov. 2013. http://wiki.apache.org/cassandra/InternodeEncryption.

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

[9] WebTrust for certification authorities — SSL baseline requirements audit criteria v.1.1, Jan. 2013. http://www.webtrust.org/homepage-documents/item72056.pdf.

[10] Android Platform Versions, Apr. 2014. https://developer.android.com/about/dashboards/index.html#Platform.

[11] Apple Says iOS, OSX and Key Web Services Not Affected by Heartbleed Security Flaw, Apr. 2014. http://recode.net/2014/04/10/apple-says-ios-osx-and-key-web-services-not-affected-by-heartbleed-security-flaw/.

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

[13] The Heartbleed Hit List: The Passwords You Need to Change Right Now, Apr. 2014. http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/.

[14] HP Support Document c04249852, May 2014. http://goo.gl/AcUG8I.

[15] Is Openfire Affected by Heartbleed?, Apr. 2014. https://community.igniterealtime.org/thread/52272.

[16] June 2014 Web Server Survey, 2014. http://news.netcraft.com/archives/2014/06/06/june-2014-web-server-survey.html.

[17] NGINX and the Heartbleed Vulnerability, Apr. 2014. http://nginx.com/blog/nginx-and-the-heartbleed-vulnerability/.

[18] Official BTCJam Update, Apr. 2014. http://blog.btcjam.com/post/82158642922/official-btcjam-update.

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

[20] Tomcat Heartbleed, Apr. 2014. https://wiki.apache.org/tomcat/Security/Heartbleed.

[21] Wikimedias Response to the "Heartbleed" Security Vulnerability, Apr. 2014. https://blog.wikimedia.org/2014/04/10/

wikimedias-response-to-the-heartbleed-security-vulnerability/.
[22] Adobe. Heartbleed Update, Apr. 2014. http://blogs.adobe.com/psirt/?p=1085.
[23] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security (CCS'15)*, 2015.
[24] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.
[25] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman. Zippier ZMap: Internet-wide scanning at 10 Gbps. In *8th USENIX Workshop on Offensive Technologies*, Aug. 2014.
[26] S. Ajmani, B. Fitzpatrick, A. Gerrand, R. Griesemer, A. Langley, R. Pike, D. Symonds, N. Tao, and I. L. Taylor. A conversation with the Go team. Golang blog, 2013. http://blog.golang.org/a-conversation-with-the-go-team.
[27] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer. Here's my cert, so trust me, maybe? Understanding TLS errors on the web. In *22nd International Conference on the World Wide Web*, 2013.
[28] D. Akhawe and A. P. Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *USENIX security symposium*, 2013.
[29] M. Al-Bassam. Top Alexa 10,000 Heartbleed Scan—April 14, 2014. https://github.com/musalbas/heartbleed-masstest/blob/94cd9b6426311f0d20539e696496ed3d7bdd2a94/top1000.txt.
[30] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt. On the security of RC4 in TLS. In *22nd USENIX Security Symposium*, Aug. 2013.
[31] Alienth. We Recommend that You Change Your Reddit Password, Apr. 2014. http://www.reddit.com/r/announcements/comments/231hl7/we_recommend_that_you_change_your_reddit_password/.
[32] B. Amann, M. Vallentin, S. Hall, and R. Sommer. Extracting Certificates from Live Traffic: A Near Real-Time SSL Notary Service. Technical Report TR-12-014, ICSI, Nov. 2012.
[33] Anonymous. Internet census 2012. http://census2012.sourceforge.net/paper.html, Mar. 2013.
[34] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX Security Symposium*, 2017.
[35] Apache. Lucene. https://lucene.apache.org.
[36] B. Arkin. Adobe important customer security announcement, Oct. 2013. http://blogs.adobe.com/conversations/2013/10/important-customer-security-announcement.html.
[37] AWeber Communications. Heartbleed: We're Not Affected. Here's What You Can Do To Protect Yourself, Apr. 2014. http://blog.aweber.com/articles-tips/

heartbleed-how-to-protect-yourself.htm.
[38] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan. The menlo report. In *IEEE Security and Privacy*, 2012.
[39] S. Banon. Elasticsearch, 2013. https://www.elastic.co.
[40] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. 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.
[41] E. Barker and A. Roginsky. 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.
[42] G. Bartlett, J. Heidemann, and C. Papadopoulos. Understanding passive and active service discovery. In *7th ACM Internet Measurement Conference*, 2007.
[43] M. Bauer and B. Laurie. Factoring silly keys from the keyservers. In *The Shoestring Foundation Weblog*, July 2004. http://shoestringfoundation.org/cgi-bin/blosxom.cgi/2004/07/01#non-pgp-key.
[44] D. Beazley. SWIG: An easy to use tool for integrating scripting languages with C and C++. In *4th USENIX Tcl/Tk Workshop*, 1996.
[45] M. Bellare, Z. Brakerski, M. Naor, T. Ristenpart, G. Segev, H. Shacham, and S. Yilek. Hedged public-key encryption: How to protect against bad randomness. In *15th Asiacrypt*, Dec. 2009.
[46] M. Bellare, S. Goldwasser, and D. Micciancio. "Pseudo-random" generators within cryptographic applications: the DSS case. In *17th Advances in Cryptology*, 1997.
[47] L. Bello. DSA-1571-1 OpenSSL—Predictable random number generator, 2008. Debian Security Advisory. http://www.debian.org/security/2008/dsa-1571.
[48] D. J. Bernstein. SYN cookies. http://cr.yp.to/syncookies.html, 1996.
[49] D. J. Bernstein. How to find the smooth parts of integers. May 2004. http://cr.yp.to/papers.html#smoothparts.
[50] D. J. Bernstein. Fast multiplication and its applications. In *Algorithmic Number Theory*, May 2008.
[51] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *36th IEEE Symposium on Security and Privacy*, May 2015.
[52] S. Bhat. Gmail users in Iran hit by MITM Attacks, Aug. 2011. http://techie-buzz.com/tech-news/gmail-iran-hit-mitm.html.
[53] Bitcoin. OpenSSL Heartbleed Vulnerability, Apr. 2014. https://bitcoin.org/en/alert/2014-04-11-heartbleed.
[54] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In *19th Advances in Cryptology*, 1999.
[55] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 3546, June 2003.
[56] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.
[57] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[58] A. Bonkoski, R. Bielawski, and J. A. Halderman. Illuminating the security issues surrounding lights-out server management. In *8th USENIX Workshop on Offensive Technologies*, 2014.

[59] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow. Elliptic curve cryptography in practice. In *18th International Conference on Financial Cryptography and Data Security*, Mar. 2014.

[60] E. Brier, C. Clavier, J. Coron, and D. Naccache. Cryptanalysis of RSA signatures with fixed-pattern padding. In *21st Advances in Cryptology*, 2001.

[61] Bro Network Security Monitor Web Site. http://www.bro.org.

[62] D. R. L. Brown. Standards for efficient cryptography 1: Elliptic curve cryptography, 2009. http://www.secg.org/download/aid-780/sec1-v2.pdf.

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

[64] bushing, marcan, segher, and sven. Console hacking 2010: PS3 epic fail. 27th Chaos Communication Congress, 2010. http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf.

[65] 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.

[66] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayerj. OpenPGP message format. RFC 4880, 2007. https://www.ietf.org/rfc/rfc4880.txt.

[67] D. Campbell. Update on security incident and additional security measures, 2015. https://sendgrid.com/blog/update-on-security-incident-and-additional-security-measures/.

[68] S. Checkoway, M. Fredrikson, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, and H. Shacham. On the practical exploitability of Dual EC in TLS implementations. In *23rd USENIX Security Symposium*, Aug. 2014.

[69] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. In *26th IEEE Symposium on Foundations of Computer Science*, 1985.

[70] T. Chown. IPv6 Implications for Network Scanning. RFC 5157, Mar. 2008.

[71] Cisco. Cisco ASA 5500-X series next-generation firewalls, 2015. http://www.cisco.com/c/en/us/products/security/asa-5500-series-next-generation-firewalls/index.html.

[72] Cisco. Cisco IOS Firewall, 2015. http://www.cisco.com/c/en/us/products/security/ios-firewall/index.html.

[73] Cisco. SMTP and ESMTP inspection overview, 2015. http://www.cisco.com/c/en/us/td/docs/security/asa/asa93/configuration/firewall/asa-firewall-cli/inspect-basic.html#pgfId-2490137.

[74] Codenomicon. Heartbleed bug, 2014. http://heartbleed.com/.

[75] L. Constantin. Yahoo email anti-spoofing policy breaks mailing lists, 2014. http://www.pcworld.com/article/2141120/yahoo-email-antispoofing-policy-breaks-mailing-lists.html.

[76] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, 2008.

[77] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[78] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, and S. Antipolis. A large scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium*, 2014.

[79] M. Cox, R. Engelschall, S. Henson, B. Laurie, et al. The OpenSSL project. http://www.openssl.org.

[80] N. Craver. Is Stack Exchange Safe from Heartbleed?, Apr. 2014. http://meta.stackexchange.com/questions/228758/is-stack-exchange-safe-from-heartbleed.

[81] D. Crocker, T. Hansen, and M. Kucherawy. DomainKeys Identified Mail (DKIM) signatures. RFC 6379, Sept. 2011. https://tools.ietf.org/html/rfc6376.

[82] D. Crocker and T. Zink. M3AAWG trust in email begins with authentication, 2015. https://www.m3aawg.org/sites/maawg/files/news/M3AAWG_Email_Authentication_Update-2015.pdf.

[83] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks. In *14th ACM Internet Measurement Conference*, Nov. 2014.

[84] D. Dagon, N. Provos, C. P. Lee, and W. Lee. Corrupted DNS resolution paths: The rise of a malicious resolution authority. In *Network and Distributed Systems Symposium*, 2008.

[85] D. Davis, R. Ihaka, and P. Fenstermacher. Cryptographic randomness from air turbulence in disk drives. In *14th Advances in Cryptology*, 1994.

[86] L. Deri. Improving passive packet capture: Beyond device polling. In *4th International System Administration and Network Engineering Conference*, 2004.

[87] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[88] R. Dingledine. Research problems: Ten ways to discover Tor bridges. http://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges, Oct. 2011.

[89] R. Dingledine. Tor OpenSSL Bug CVE-2014-0160, Apr. 2014. https://blog.torproject.org/blog/openssl-bug-cve-2014-0160.

[90] L. Dorrendorf, Z. Gutterman, and B. Pinkas. Cryptanalysis of the Windows random number generator. In *14th ACM Conference on Computer and Communications Security*, 2007.

[91] Dropbox Support. https://twitter.com/dropbox_support/status/453673783480832000, Apr. 2014. Quick Update on Heartbleed: We've Patched All of Our User-Facing Services & Will Continue to Work to Make Sure Your Stuff is Always Safe.

[92] H. Duan, N. Weaver, Z. Zhao, M. Hu, J. Liang, J. Jiang, K. Li, and V. Paxson. Hold-on: Protecting against on-path DNS poisoning. In *Workshop on Securing and Trusting Internet Names*, 2012.

[93] V. Dukhovni and W. Hardaker. SMTP security via opportunistic DANE TLS, July 2013. http://tools.ietf.org/html/draft-ietf-dane-smtp-with-dane-12.

[94] Z. Durumeric, D. Adrian, M. Bailey, and J. A. Halderman. Heartbleed bug health

report, 2014. https://zmap.io/heartbleed/.

[95] Z. Durumeric, D. Adrian, J. Kasten, D. Springall, M. Bailey, and J. A. Halderman. POODLE attack and SSLv3 deployment, 2014. https://poodle.io.

[96] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. Censys: A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, 2015.

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

[98] Z. Durumeric, M. Bailey, and J. A. Halderman. An Internet-wide view of Internet-wide scanning. In *23rd USENIX Security Symposium*, Aug. 2014.

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

[100] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The matter of Heartbleed. In *14th ACM Internet Measurement Conference*, 2014.

[101] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*, Aug. 2013.

[102] P. Eckersley and J. Burns. An observatory for the SSLiverse. Talk at Defcon 18 (2010). https://www.eff.org/files/DefconSSLiverse.pdf.

[103] A. Ellis. Akamai heartbleed Update (V3), Apr. 2014. https://blogs.akamai.com/2014/04/heartbleed-update-v3.html.

[104] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP. RFC 7469, 2015. http://tools.ietf.org/html/rfc7469.

[105] Facebook. RocksDB: A persistent key-value store for fast storage environments. http://rocksdb.org/.

[106] Facebook. The current state of SMTP STARTTLS deployment, May 2014. https://www.facebook.com/notes/protect-the-graph/the-current-state-of-smtp-starttls-deployment/1453015901605223/.

[107] Facebook. Massive growth in SMTP STARTTLS deployment, Aug. 2014. https://www.facebook.com/notes/protect-the-graph/massive-growth-in-smtp-starttls-deployment/1491049534468526.

[108] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving SSL warnings: Comprehension and adherence. In *33rd ACM Conference on Human Factors in Computing Systems*, 2015.

[109] I. Foster, J. Larson, M. Masich, A. Snoeren, S. Savage, and K. Levchenko. Security by any other name: On the effectiveness of provider based email security. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[110] A. S. Foundation. CouchDB and the Heartbleed SSL/TLS Vulnerability, Apr. 2014. https://blogs.apache.org/couchdb/entry/couchdb_and_the_heartbleed_ssl.

[111] N. T. Foundation. Open NTP project. http://openntpproject.org/.

[112] R. Getz. IRQF_SAMPLE_RANDOM question ... Linux Kernel Mailing List post. https://lkml.org/lkml/2009/4/6/283.

[113] GoDaddy. OpenSSL Heartbleed: We've Patched Our Servers, Apr. 2014. http:

//support.godaddy.com/godaddy/openssl-and-heartbleed-vulnerabilities/.
[114] I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr. Dobb's Journal*, 21(1):66–70, 1996.
[115] Golden Frog. The FCC must prevent ISPs from blocking encryption. http://www.goldenfrog.com/blog/fcc-must-prevent-isps-blocking-encryption.
[116] Google. Protocol buffers. https://github.com/google/protobuf.
[117] R. Graham. MASSCAN: Mass IP port scanner. https://github.com/robertdavidgraham/masscan.
[118] L. Grangeia. Heartbleed, Cupid and Wireless, May 2014. http://www.sysvalue.com/en/heartbleed-cupid-wireless/.
[119] T. Granlund et al. The GNU multiple precision arithmetic library. http://gmplib.org/.
[120] S. Grant. 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.
[121] B. Grubb. Heartbleed Disclosure Timeline: Who Knew What and When. Apr. 2014. http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html.
[122] X. Gu and X. Gu. On the detection of fake certificates via attribute correlation. *Entropy*, 17(6), 2015.
[123] P. Gutmann. Software generation of random numbers for cryptographic purposes. In *7th USENIX Security Symposium*, 1998.
[124] P. Gutmann. Lessons learned in implementing and deploying crypto software. In *11th USENIX Security Symposium*, 2002.
[125] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the Linux random number generator. In *IEEE Symposium on Security and Privacy*, May 2006.
[126] L. Haisley. OpenSSL Crash with STARTTLS in Courier, May 2014. http://sourceforge.net/p/courier/mailman/message/32298514/.
[127] J. A. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. Lest we remember: Cold boot attacks on encryption keys. In *17th USENIX Security Symposium*, July 2008.
[128] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated software router. In *ACM SIGCOMM*, Sept. 2010.
[129] C. Heffner et al. LittleBlackBox: Database of private SSL/SSH keys for embedded devices. http://code.google.com/p/littleblackbox/.
[130] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister. Census and survey of the visible Internet. In *8th ACM Internet Measurement Conference*, 2008.
[131] J. Heidemann, L. Quan, and Y. Pradkin. A preliminary analysis of network outages during hurricane sandy. Technical Report ISI-TR-2008-685b, USC/Information Sciences Institute, November 2012.
[132] N. Heninger. Factoring as a service. Rump session talk, *Crypto* 2013.
[133] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium*, 2012.
[134] P. Hoffman. SMTP service extension for secure SMTP over transport layer security.

RFC 3207, Feb. 2002. http://www.ietf.org/rfc/rfc3207.txt.
[135] J. Hoffman-Andrews. Isps removing their customers' email encryption. https://www.eff.org/deeplinks/2014/11/starttls-downgrade-attacks.
[136] J. Hoffman-Andrews and P. Eckersley. STARTTLS everywhere, June 2014. https://github.com/EFForg/starttls-everywhere.
[137] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: A thorough analysis of the X.509 PKI using active and passive measurements. In *11th ACM Internet Measurement Conference*, 2011.
[138] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. (2459), Jan. 2009.
[139] N. Howgrave-Graham and N. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.
[140] L.-S. Huang, S. Adhikarla, D. Boneh, and C. Jackson. An experimental study of TLS forward secrecy deployments. In *Web 2.0 Security and Privacy (W2SP)*, 2014.
[141] IBM. OpenSSL Heartbleed (CVE-2014-0160), May 2014. https://www-304.ibm.com/connections/blogs/PSIRT/entry/openssl_heartbleed_cve_2014_0160.
[142] Infusionsoft. What You Need to Know About Heartbleed, Apr. 2014. http://blog.infusionsoft.com/company-news/need-know-heartbleed/.
[143] Internal Revenue Service. IRS Statement on "Heartbleed" and Filing Season, Apr. 2014. http://www.irs.gov/uac/Newsroom/IRS-Statement-on-Heartbleed-and-Filing-Season.
[144] V. Jacobson, C. Leres, and S. McCanne. libpcap. Lawrence Berkeley National Laboratory, 1994.
[145] W. Kamishlian and R. Norris. Installing OpenSSL for Jabberd 2. http://www.jabberdoc.org/app_openssl.html.
[146] M. Karami, Y. Park, and D. McCoy. Stress testing the booters: understanding and undermining the business of DDoS services. In *25th International Conference on World Wide Web*, 2016.
[147] J. Kasten, E. Wustrow, and J. A. Halderman. Cage: Taming certificate authorities by inferring restricted scopes. In *17th International Conference on Financial Cryptography and Data Security (FC)*, 2013.
[148] K. Kinder. Event-driven programming with Twisted and Python. *Linux Journal*, 2005(131):6, 2005.
[149] S. Kitterman. Sender policy framework (SPF) for authorizing use of domains in email. RFC 7208, Apr. 2014. http://tools.ietf.org/html/rfc7208.
[150] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, et al. Factorization of a 768-bit rsa modulus. In *30th Advances in Cryptology*. 2010.
[151] J. Klensin. Simple mail transfer protocol. RFC 5321, Oct. 2008. http://tools.ietf.org/html/rfc5321.
[152] M. Koster. A standard for robot exclusion. http://www.robotstxt.org/orig.html, 1994.
[153] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Network and Distributed System Security Symposium*, 2015.
[154] M. Kucherawy and E. Zwicky. Domain-based message authentication, reporting, and

conformance (DMARC). RFC 7489, Mar. 2015. https://tools.ietf.org/html/rfc7489.

[155] M. Kührer, T. Hupperich, C. Rossow, and T. Holz. Exit from hell? reducing the impact of amplification DDoS attacks. In *23rd USENIX Security Symposium*, 2014.

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

[157] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency, 2013. http://www.certificate-transparency.org/.

[158] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. (6962), 2013.

[159] E. Law. Understanding certificate name mismatches. http://blogs.msdn.com/b/ieinternals/archive/2009/12/07/certificate-name-mismatch-warnings-and-server-name-indication.aspx, 2009.

[160] N. Lawson. DSA requirements for random k value, 2010. http://rdist.root.org/2010/11/19/dsa-requirements-for-random-k-value/.

[161] A. Lenstra, H. Lenstra, M. Manasse, and J. Pollard. The number field sieve. In *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. 1993.

[162] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, Whit is right. *IACR Cryptology ePrint Archive*, 2012:64, 2012.

[163] D. Leonard and D. Loguinov. Demystifying service discovery: Implementing an Internet-wide scanner. In *10th ACM Internet Measurement Conference*, 2010.

[164] Litespeed Technologies. LSWS 4.2.9 Patches Heartbleed Bug, Apr. 2014. http://www.litespeedtech.com/support/forum/threads/lsws-4-2-9-patches-heartbleed-bug.8504/.

[165] Y. Liu, A. Sarabi, J. Zhang, P. Naghizadeh, M. Karir, M. Bailey, and M. Liu. Cloudy with a chance of breach: Forecasting cyber security incidents. In *24th USENIX Security Symposium*, Aug. 2015.

[166] G. Locke and P. Gallagher. FIPS PUB 186-3: Digital Signature Standard (DSS). Federal Information Processing Standards Publication (2009).

[167] G. Lowe, P. Winters, and M. L. Marcus. The great DNS wall of China. Technical report, New York University, 2007. http://cs.nyu.edu/~pcw216/work/nds/final.pdf.

[168] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[169] W. R. Marczak, J. Scott-Railton, M. Marquis-Boire, and V. Paxson. When governments hack opponents: A look at actors and technology. In *23rd USENIX Security Symposium*, 2014.

[170] M. Marlinspike. SSL and the future of authenticity, Aug. 2011. Talk at BlackHat 2011. http://www.thoughtcrime.org/blog/ssl-and-the-future-of-authenticity/.

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

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

[173] J. Matherly. Shodan FAQ. http://www.shodanhq.com/help/faq.
[174] N. Mathewson and N. Provos. libevent—An event notification library. http://libevent.org.
[175] J. Mauch. Open resolver project. http://openresolverproject.org/.
[176] A. May and M. Ritzenhofen. Implicit factoring: On polynomial time factoring given only an implicit hint. In *Public Key Cryptography*, 2009.
[177] N. Mehta and Codenomicon. The heartbleed bug. http://heartbleed.com/.
[178] Microsoft. Microsoft Services unaffected by OpenSSL Heartbleed vulnerability, Apr. 2014. http://blogs.technet.com/b/security/archive/2014/04/10/microsoft-devices-and-services-and-the-openssl-heartbleed-vulnerability.aspx.
[179] Microsoft. TLS functionality and related terminology, June 2014. http://technet.microsoft.com/en-us/library/bb430753%28v=exchg.150%29.aspx.
[180] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujit, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey. An Internet-wide view of ICS devices. In *14th IEEE Privacy, Security, and Trust Conference (PST'16)*, 2016.
[181] I. Mironov. Factoring RSA moduli. Part II, 2012. http://windowsontheory.org/2012/05/17/factoring-rsa-moduli-part-ii/.
[182] Modbus IDA. MODBUS TCP implementation guide, Oct. 2006. http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.
[183] B. Moller, T. Duong, and K. Kotowicz. This POODLE bites: Exploiting the SSL 3.0 fallback, 2014. https://www.openssl.org/~bodo/ssl-poodle.pdf.
[184] MongoDB. MongoDB Response on Heartbleed OpenSSL Vulnerability, Apr. 2014. http://www.mongodb.com/blog/post/mongodb-response-heartbleed-openssl-vulnerability.
[185] H. Moore. Security flaws in universal plug and play. Unplug. Don't Play, Jan. 2013. http://community.rapid7.com/servlet/JiveServlet/download/2150-1-16596/SecurityFlawsUPnP.pdf.
[186] K. Murchison. Heartbleed Warning - Cyrus Admin Passowrd Leak!, Apr. 2014. http://lists.andrew.cmu.edu/pipermail/info-cyrus/2014-April/037351.html.
[187] Z. Nabi. The anatomy of web censorship in Pakistan. *arXiv:1307.1144*, 2013.
[188] Netcraft, Ltd. Web server survey. http://news.netcraft.com/archives/2013/05/03/may-2013-web-server-survey.html, May 2013.
[189] NetMarketShare. Desktop operating system market share, Apr. 2013. http://www.netmarketshare.com/operating-system-market-share.aspx.
[190] M. D. Network. Mozilla network security services (nss). http://www.mozilla.org/projects/security/pki/nss/.
[191] E. Ng. Tunnel Fails after OpenSSL Patch, Apr. 2014. https://lists.openswan.org/pipermail/users/2014-April/022934.html.
[192] J. Nightingale. Comodo Certificate Issue – Follow Up, Mar. 2011. https://blog.mozilla.org/security/2011/03/25/comodo-certificate-issue-follow-up/.
[193] M. O'Connor. Google Services Updated to Address OpenSSL CVE-2014-0160 (the Heartbleed Bug), Apr. 2014. http://googleonlinesecurity.blogspot.com/2014/04/google-services-updated-to-address.html.
[194] P. Ondruska. Does OpenSSL CVE-2014-0160 Effect Jetty Users?, Apr. 2014. http://dev.eclipse.org/mhonarc/lists/jetty-users/msg04624.html.

[195] OpenSSL Project Team. OpenSSL Security Advisory, Apr. 2014. http://www.mail-archive.com/openssl-users@openssl.org/msg73408.html.
[196] OpenSSL Project Team. OpenSSL Version 1.0.1g Released, Apr. 2014. http://www.mail-archive.com/openssl-users@openssl.org/msg73407.html.
[197] OpenVPN. OpenSSL Vulnerability—Heartbleed. https://community.openvpn.net/openvpn/wiki/heartbleed.
[198] Oracle. OpenSSL Security Bug—Heartbleed / CVE-2014-0160, Apr. 2014. http://www.oracle.com/technetwork/topics/security/opensslheartbleedcve-2014-0160-2188454.html.
[199] L. Padron. Important Read Critical Security Advisory And Patch for OpenSSL Heartbleed Vulnerability, Apr. 2014. http://blog.zimbra.com/blog/archives/2014/04/important-read-critical-security-advisory-patch-openssl-heartbleed-vulnerability.html.
[200] C. Palmer and R. Sleevi. Gradually sunsetting SHA-1. Google Online Security Blog. http://googleonlinesecurity.blogspot.com/2014/09/gradually-sunsetting-sha-1.html.
[201] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.
[202] PayPal. OpenSSL Heartbleed Bug—PayPal Account Holders are Secure, Apr. 2014. https://www.paypal-community.com/t5/PayPal-Forward/OpenSSL-Heartbleed-Bug-PayPal-Account-Holders-are-Secure/ba-p/797568.
[203] R. Pike. Concurrency is not parallelism. In *Heroku Waza*, Jan. 2012. http://talks.golang.org/2012/waza.slide.
[204] W. Pinckaers. http://lekkertech.net/akamai.txt.
[205] J. B. Postel. Simple mail transfer protocol. RFC 821, Aug. 1982.
[206] http://www.postfix.org/postconf.5.html#smtp_tls_security_level.
[207] M. Prince. The Hidden Costs of Heartbleed, Apr. 2014. http://blog.cloudflare.com/the-hard-costs-of-heartbleed.
[208] N. Provos and P. Honeyman. ScanSSH: Scanning the Internet for SSH servers. In *16th USENIX Systems Administration Conference (LISA)*, 2001.
[209] Publishers Clearing House. Stay Smart About The "Heartbleed" Bug With PCH!, Apr. 2014. http://blog.pch.com/blog/2014/04/16/stay-smart-about-the-heartbleed-bug-with-pch/.
[210] Rackspace. Protect Your Systems From "Heartbleed" OpenSSL Vulnerability, Apr. 2014. http://www.rackspace.com/blog/protect-your-systems-from-heartbleed-openssl-vulnerability/.
[211] B. Ramsdell and S. Turner. Secure/multipurpose Internet mail extensions (S/MIME) version 3.2 message specification. RFC 5751. https://tools.ietf.org/html/rfc5751.
[212] Red Hat. How to Recover from the Heartbleed OpenSSL vulnerability, Apr. 2014. https://access.redhat.com/articles/786463.
[213] E. Rescorla. *SSL and TLS: designing and building secure systems*, volume 1. Addison-Wesley Reading, 2001.
[214] R. Richmond. Comodo fraud incident, Mar. 2011. http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html.
[215] S. Rijs and M. van der Meer. The state of StartTLS, June 2014. https://caldav.os3.nl/_media/2013-2014/courses/ot/magiel_sean2.pdf.

[216] T. Ristenpart and S. Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Network and Distributed Security Symposium*, 2010.

[217] I. Ristic. 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.

[218] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21, Feb. 1978.

[219] L. Rizzo. netmap: A novel framework for fast packet I/O. In *2012 USENIX Annual Technical Conference*, 2012.

[220] S. Sanfilippo and P. Noordhuis. Redis. http://redis.io.

[221] T. Saunders. ProFTPD and the OpenSSL "Heartbleed" Bug, May 2014. http://comments.gmane.org/gmane.network.proftpd.user/9465.

[222] B. Say. Bleedingheart Bug in OpenSSL, Apr. 2014. http://www.stunnel.org/pipermail/stunnel-users/2014-April/004578.html.

[223] H. Scholz. SIP stack fingerprinting and stack difference attacks. Talk at Blackhat 2006. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Scholz.pdf.

[224] A. Schulman and N. Spring. Pingin' in the rain. In *11th ACM Internet Measurement Conference*, 2011.

[225] R. Seggelmann, M. Tuexen, and M. Williams. 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.

[226] N. Siong and H. Toivonen. M2Crypto Python interface to OpenSSL. http://sandbox.rulemaker.net/ngps/m2/.

[227] K. Sklower. A tree-based packet routing table for Berkeley Unix. In *Winter USENIX Conference*, 1991.

[228] M. Smith and D. Loguinov. Enabling high-performance Internet-wide measurements on Windows. In *11th International Conference on Passive and Active Measurement*, 2010.

[229] Solid IT. DB-Engines ranking. http://db-engines.com/en/ranking.

[230] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today. http://www.win.tue.nl/hashclash/rogue-ca/.

[231] D. Springall, Z. Durumeric, and J. A. Halderman. FTP: The forgotten cloud. In *IEEE/IFIP Conference on Dependable Systems and Networks (DSN'16)*, 2016.

[232] D. Springall, Z. Durumeric, and J. A. Halderman. Measuring the security harm of TLS crypto shortcuts. In *16th ACM Internet Measurement Conference*, 2016.

[233] W. R. Stevens and G. R. Wright. *TCP/IP Illustrated: The Implementation*, volume 2. 1995.

[234] N. Sullivan. The Results of the CloudFlare Challenge. Apr. 2014. http://blog.cloudflare.com/the-results-of-the-cloudflare-challenge.

[235] G. Taylor and G. Cox. Behind Intel's new random-number generator. *IEEE Spectrum*, Sept. 2011.

[236] Telecom Asia. Google, Yahoo SMTP email severs hit in thailand. http://www.telecomasia.net/content/google-yahoo-smtp-email-severs-hit-thailand.

[237] C. Tokunaga, D. Blaauw, and T. Mudge. True random number generator with a metastability-based quality control. *IEEE Journal of Solid-State Circuits*, 2008.
[238] Tor Project. Tor Bridges. https://www.torproject.org/docs/bridges, 2008.
[239] Tor Project. obfsproxy. https://www.torproject.org/projects/obfsproxy.html.en, 2012.
[240] Tumblr. Urgent Security Update, Apr. 2014. http://staff.tumblr.com/post/82113034874/urgent-security-update.
[241] United States Postal Service. Avoiding Heartbleed, May 2014. https://ribbs.usps.gov/importantupdates/HeartbleedArticle.pdf.
[242] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman. Towards a complete view of the certificate ecosystem. In *16th ACM Internet Measurement Conference*, 2016.
[243] M. Vanhoef and F. Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *24th USENIX Security Symposium*, Aug. 2015.
[244] Verisign Labs. DNSSEC scoreboard, 2015. http://scoreboard.verisignlabs.com/.
[245] J.-P. Verkamp and M. Gupta. Inferring mechanics of web censorship around the world. *3rd USENIX Workshop on Free and Open Communications on the Internet*, Aug. 2012.
[246] J. Viega, M. Messier, and P. Chandra. *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly, 2002.
[247] M. Vincent. TI-83 Plus OS signing key cracked. In *ticalc.org weblog*, July 2009. http://www.ticalc.org/archives/news/articles/14/145/145154.html.
[248] T. Wilde. Great Firewall Tor probing. https://gist.github.com/twilde/da3c7a9af01d74cd7de7, 2012.
[249] K. Wilson. Bug 523652 - IPS action items re IPS SERVIDORES root certificate, Nov. 2009. https://bugzilla.mozilla.org/show_bug.cgi?id=523652.
[250] M. Wimmer. Removed Support for OpenSSL, 2007. https://jabberd.org/hg/amessagingd/rev/bcb8eb80cbb9.
[251] R. Woolley, M. Murray, M. Dounin, and R. Ermilov. FreeBSD security advisory FreeBSD-SA-08:11.arc4random, 2008. http://lists.freebsd.org/pipermail/freebsd-security-notifications/2008-November/000117.html.
[252] WordPress. Heartbleed Security Update, Apr. 2014. http://en.blog.wordpress.com/2014/04/15/security-update/.
[253] E. Wustrow, C. Swanson, and J. A. Halderman. TapDance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium*, 2014.
[254] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In *9th ACM Internet Measurement Conference*.
[255] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When Private Keys Are Public: Results from the 2008 Debian OpenSSL Vulnerability. In *Proc. ACM Internet Measurement Conference*, Nov. 2009.
[256] T. Ylonen. SSH—secure login connections over the internet. In *6th USENIX Security Symposium*, 1996.
[257] T. Ylönen and C. Lonvick. The secure shell (SSH) protocol architecture. 2006. http://merlot.tools.ietf.org/html/rfc4251.
[258] ZDNet. Heartbleed Bug Affects Yahoo, OKCupid Sites, Apr. 2014. http://www.zdnet.

com/heartbleed-bug-affects-yahoo-imgur-okcupid-convo-7000028213/.
[259] ZEDOinc. https://twitter.com/ZEDOinc/status/456145140503957504, Apr. 2014. Customers and partners: none of the ZEDO sites or assets are affected by Heartbleed.
[260] K. Zetter. A cyberattack has caused confirmed physical damage for the second time ever, 2015. http://www.wired.com/2015/01/german-steel-mill-hack-destruction/.