Towards Certificate Transparency Log Monitoring

Harshal Sheth

under the direction of
Samuel Erb and Tom Houman
Akamai Technologies

Research Science Institute
October 28, 2017

Abstract

After many incidents related to compromised Certificate Authorities (CAs) and misissued certificates, the proposed Certificate Transparency (CT) protocol brings SSL certificates into publicly-inspectable certificate logs. The responsibility of cryptographically verifying the integrity of these logs and scanning for suspicious certificates falls to the certificate monitors. Over 120 million certificates contained within such logs were analyzed to understand security trends and metrics within the SSL ecosystem. In addition, these certificates were examined with the end-goal of identifying suspicious certificates, as a certificate monitor would do. A number of certificates were identified as violating the CA/B Forum Baseline Requirements and using weak cryptographic techniques such as MD5 and 512-bit RSA keys.

Summary

Encryption on the Internet uses SSL certificates to verify the identities of web servers and prevent spoofing attacks. Such certificates are issued by well-known and trusted third parties known as Certificate Authorities (CAs). In response to the increasing frequency of CA-related security incidents, the recently developed Certificate Transparency project allows the public to monitor and audit the activities of CAs. This paper uses Certificate Transparency to conduct an overall survey of the SSL certificate ecosystem, as well as identifies a number of certificates that are either cryptographically weak or in violation of CA standards.

1 Introduction

The Internet is inherently insecure: any party can read, modify, or spoof any communication at any point during its transmission. To preserve privacy, the Hypertext Transfer Protocol Secure (HTTPS) is used. According to data collected via the Chrome and Firefox browsers, not all Web servers utilize HTTPS, and thus between 30 and 50 percent of web traffic is still unencrypted [1, 2]. This insecurity has led to several initiatives, backed by Google, Mozilla, and the U.S. government alike, towards a fully encrypted Web [3, 4].

Merely encrypting transmitted data is not sufficient; an attacker can masquerade as the destination server, and thus render the encryption inefficacious. To solve this problem, Certificate Authorities (CAs) perform identity verification on Web servers and their operating organizations, and issue to them digitally signed, tamper-proof *certificates* which assert that such identity verification took place. However, CAs are subject to limited oversight [5], and thus there have been many instances in which certificates were misissued or the CA was compromised. The Certificate Transparency (CT) proposal, which has gained significant traction, requires that these certificates be placed in publicly accessible logs. Certificate monitors can then watch these logs for suspicious certificates or trends.

In past "comprehensive" SSL certificate surveys, a corpus of certificates was obtained using domains lists and IPv4 address space scanning [6, 7]. Because Certificate Transparency logs are more expansive than these previous datasets, a more accurate survey was conducted. The results of this survey are valuable for understanding overall trends within the SSL ecosystem and allowing security researchers to pinpoint and mitigate security issues. In addition, a careful analysis of individual certificates was performed, identifying a number of weak and suspicious certificates that could otherwise endanger Internet infrastructure and users.

2 Background

2.1 HTTPS and TLS

The HTTPS protocol provides a layer of encryption by combining Transport Layer Security (TLS; sometimes referred to as SSL) with HTTP [8]. It relies on *public key cryptography* to establish a secure connection between client and Web server, which prevents an eavesdropping third-party from being able to read or modify the contents of the subsequent data exchange. However, such cryptographic techniques cannot prevent an attacker from spoofing a client or Web server. Figure 1 shows how an attacker could impersonate a user's bank to siphon off credentials, steal sensitive information, or perform actions on the user's behalf. Spoofing attacks are possible with basic public key cryptography due to the lack of identity verification, which enables an attacker to masquerade as any server.

2.2 SSL Certificates

Digital certificates solve this spoofing vulnerability by serving as a form of Web server identity verification, as well as providing cryptographic proof that a trusted authority performed this verification. The SSL certificates used on the Web follow the X.509v3 format, as specified by RFC 5280 in 2008 [9]. Each certificate contains information about the *subject* of the certificate, including a listing of domain names that it applies to. The subject proves

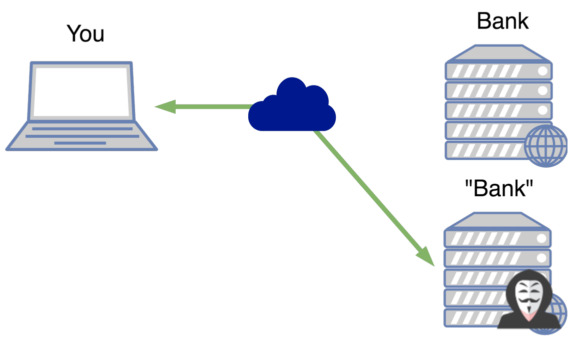

Figure 1: A server spoofing attack, in which an attacker pretends to be the intended destination of the Internet traffic.

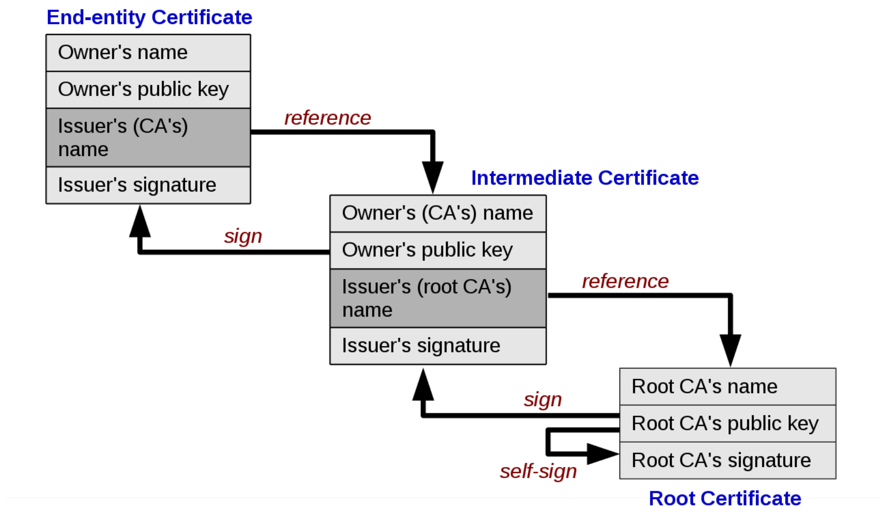

Figure 2: The chain of trust from a leaf certificate to a root certificate via an intermediate CA's certificate. Source: [10].

ownership of the certificate by remaining the sole owner of the private key corresponding to the public key that is embedded within the certificate. The information contained within a certificate is verified by a *trusted third party* called a Certificate Authority (CA). Upon verifying the identity of the subject, the CA digitally *signs* the certificate using its own certificate's private key and returns the signed certificate to the subject. The signature ensures that the certificate can be cryptographically traced back to its issuer, and additionally prevents the certificate from being modified or tampered with. Appendix A describes the digital signature mechanism in more detail.

2.2.1 Certificate Trust and Revocation

Browsers and operating systems are preloaded with a list of *root* certificates that they trust by default. Based on the trust in these root certificates, any *intermediate* certificates signed by those roots are also inherently trusted. Finally, *leaf* certificates are trusted if their signature chains lead up to a valid root certificate. As shown in Figure 2, this creates a chain of trust based on the root certificates. If any actor within this chain of trust misuses its certificate, the entire system is vulnerable. Because the trust is centralized in a relatively

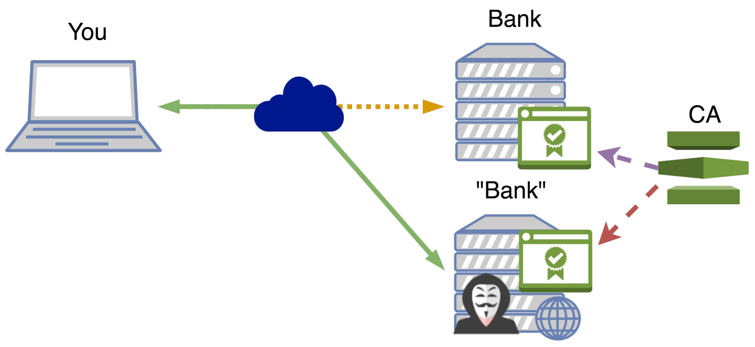

Figure 3: An attacker with a misissued certificate from a legitimate CA can still spoof a host and hijack seemingly secure connections.

limited number of CAs, the damages caused when such trust is broken is widespread.

This system puts an unreasonable amount of trust in CAs, especially when there have been a number of cases in which CAs have acted against this trust, either intentionally or unintentionally [11]. A single compromised CA presents severe dangers to all Internet users: an attacker in possession of a compromised CA certificate or misissued certificate can use it for a variety of malicious purposes (Figure 3). Furthermore, it can take weeks or months before this situation is detected and the certificates in question are revoked [12].

2.3 Certificate Transparency

The Certificate Transparency (CT) project was started to remedy this lack of accountability. Certificate Transparency allows logs of CA-issued certificates to be publicly available so that they can be audited and monitored by the general public. This allows faster detection of suspicious certificates and gives domain owners a chance to verify that no certificates have been misissued for their domain. It also forces CAs to report publicly and be held accountable for the certificates that they issue. Google's Chrome browser currently requires that all Extended Validation certificates issued after Jan. 1, 2015 be CT compliant [13], and will require CT for all certificates beginning in April 2018 [14]. This way, certificates that a compromised CA issues will either be rejected because they are not contained in a log, or spotted and revoked because of the public monitors and auditors who watch these logs.

2.3.1 Certificate Monitors

Certificate logs are required to be append-only, and certificates can not be removed, modified, or inserted into the log at any position other than the tail-end. This requirement is made possible using Merkel trees, which provide a mechanism for cryptographically assuring that logs do not violate these requirements. The purpose of certificate monitors and auditors is to use those cryptographic mechanisms to ensure that the log servers do not misbehave and that the Merkel tree remains consistent. In addition to verifying log integrity, certificate monitors retain full copies of the entire certificate log and must watch for suspicious certificates. However, the markers of suspicious certificates are not well-defined, as they vary between situations and permute frequently.

2.4 Purpose

The rising prevalence of Certificate Transparency, and the corresponding growth in certificate logs, allows for more accurate overall evaluations of Internet HTTPS. Improvements in the comprehensiveness of such surveys allow for more clearly actionable recommendations for the security community as a whole. In addition, the process of identifying specific certificates as suspicious not only reveals information about issuing practices but also more clearly defines some markers of suspicious certificates for use in certificate monitors. This research presents a high-level survey of the SSL certificate ecosystem, as well as identifies a number of certificates with peculiar characteristics.

3 Methods

At a high level, the methodology consisted of downloading hundreds of millions of certificates from certificate logs, extracting information from these certificates, and then performing a number of different analyses on these data. These analyses were done with the purposes of (1) gaining a general understanding of the certificates being issued and (2) detecting suspicious certificates within the certificate logs.

3.1 Obtaining Data

There are a number of different certificate logs available online, each with its own purpose, limitations, and restrictions [15]. The Google Pilot log was selected for this analysis because it is the longest operating and largest log available [16] and does not have any major restrictions on the certificates allowed in the log. Because most certificates are submitted to multiple logs, using multiple logs in aggregate would be mostly redundant.

The process of downloading certificates from the certificate log was accomplished using the *ct-tools* project [17]. Some minor modifications were made to the scripts within this repository to update them to current standards and file formats. The Google Pilot certificate logs contained over 120 million certificates, and the download process spanned over a few days. Relevant information was extracted from these certificates using the Python Cryptography module [18] and compiled into a single file.

3.2 Certificate Analysis

The certificate analysis consisted of two main parts. In the first, general summary statistics and graphs were compiled over the entire dataset. In the second, individual certificates that seemed suspicious or had interesting characteristics were examined in further detail.

3.2.1 General Certificate Survey

Based on the commonly used and required fields within certificates, the following general statistics were calculated using this certificate log:

- Certificate issuers and the number of certificates they issued.
- Certificate lifetime (as measured by the difference between the `not_after` and `not_before`

dates) in relation to the year of issuance[1].

- Trends in cryptographic algorithms and techniques over time.
- SAN Count: the number of domain names the certificate is allowed to identify, as specified within the *Subject Alternative Name* (SAN) field.

3.2.2 Suspicious Certificates

While downloading and processing the certificates, a number of them caused the Python scripts to crash or throw exceptions. They were marked as suspicious, and the reasons for these crashes were investigated further using a separate utility called *certlint* [19], which examines certificate files and checks for X.509 formatting and encoding errors.

In addition, a few certificates were noteworthy in their key usage restrictions, signature algorithms, key sizes, and other features, especially in relation to the CA/Browser Forum's Baseline Requirements for CAs [20]. These certificates were inspected using the OpenSSL [21] command line utility and Comodo's online certificate search tool `crt.sh` [22].

4 Results

Ninety-four percent of all CAs, by volume, have agreed to make use of CT in their Extended Validation certificates [13]. Because not all CAs contribute all of their certificates to logs, these results may be slightly skewed towards the CAs who have already adopted CT and contribute to the Google Pilot log.

In addition, there were eight certificates with a `not_before` date in 2018, and this field was used to estimate the date of certificate issuance. Thus, most of these time-series data that was collected contain points or columns for 2018 certificates, even though they were not, in actuality, issued in 2018.

[1]Because it is not possible to precisely determine the date of issuance of a given certificate, this date was approximated using the certificate's `not_before` date.

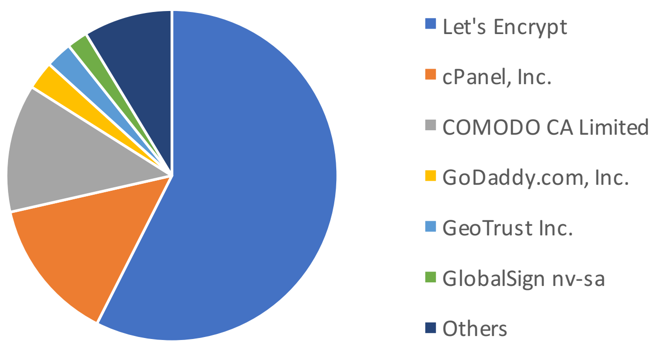

Figure 4: The issuers of the certificates in the Google Pilot log.

4.1 General Certificate Survey

Certificate Issuer The share of certificates issued by each CA is shown in Figure 4. The largest issuer, Let's Encrypt, is a relatively new, fully automated, and free CA that has issued over 100 million certificates since its inception [23]. Let's Encrypt uses the same intermediate certificate for all of its issued certificates, and thus in the case that this intermediate was compromised and revoked, tens of millions of certificates would become invalid.

Certificate Lifetime Certificates also have a set validity duration, defined by the difference between their `not_after` and `not_before` dates. Certificate lifetimes generally have been decreasing over time, although not steadily (Figure 5). In 2016 and 2017, these durations have decreased further because of the sheer quantity of 3-month Let's Encrypt certificates that flooded the Google Pilot log. This decreasing trend briefly reversed from 2012 to 2014, where the median certificate validity length did not decrease, and thus did not follow the overall trend. This is an encouraging trend: certificates with shorter lifetimes are generally less likely to be compromised and would have a lesser impact if compromised or misissued. Short-lived certificates are especially relevant because many browsers do not check certificates for revocation.

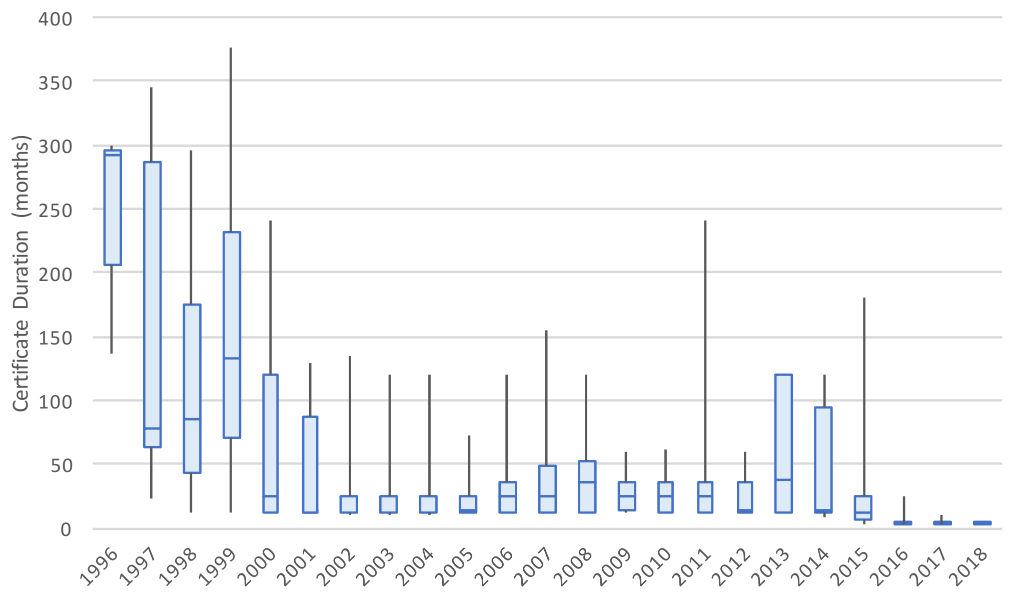

Figure 5: Time-series box plots of certificate lifetimes. To limit the effect of extreme outliers, the minimum and maximum values were replaced with the 5th and 95th percentiles.

Signature Algorithms The signature algorithms used in certificates have also evolved over time (Figure 6). The signature algorithm is a combination of the hashing algorithm (MD5, SHA1, or SHA256/384/512) and signature public key algorithm (RSA or ECDSA). After a number of cryptanalysis-based papers were published on the topic of MD5, including one in which a rogue CA was created based on a live root certificate [24], the MD5 digest algorithm was quickly abandoned for SHA1. Hash collisions have also been found for SHA1 [25], and thus hash functions from the SHA2 family (SHA256/384/512) are becoming the new standard. The transition from MD5 to SHA1 was initially slow and was completed in 2009, even though the first X.509 certificate MD5 collision was discovered by Lenstra et al. in 2005 [26]. The deprecation of SHA1 was handled quicker by CAs and was nearly complete before the first SHA1 collision was published in 2017. It can also be seen that ECDSA is becoming more popular as public cryptanalysis on RSA becomes more advanced. One notable feature of these signature algorithm trends is that there are eight certificates issued with a `not_before` date in 2018 and use the `sha1WithRSAEncryption` signature algorithm. These eight certificates are explored further in Section 4.2.2.

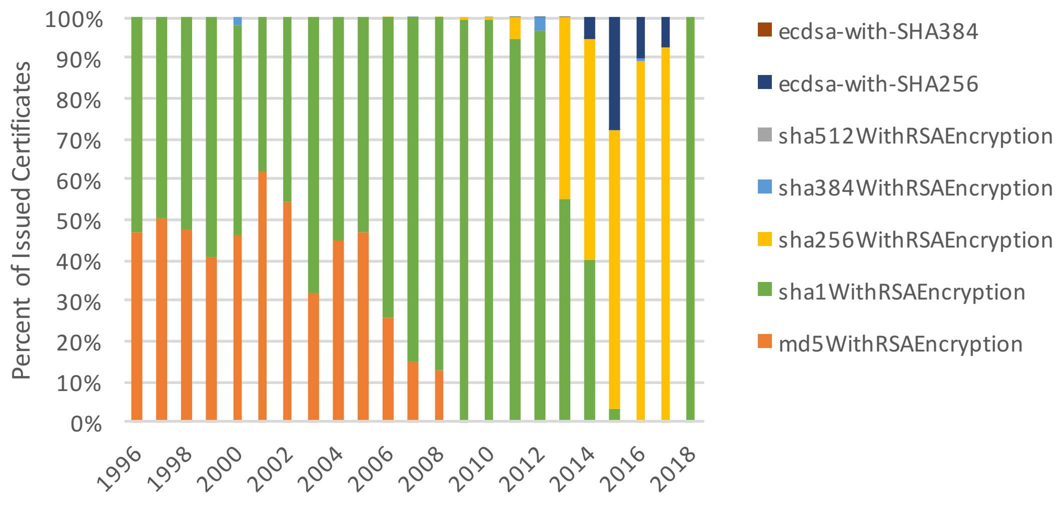

Figure 6: Signature algorithm usage over time.

SAN Count A single certificate can be used to identify multiple different names. This is accomplished using the Subject Alternative Name (SAN) field; this field lists all names under which a specific certificate is valid. The SAN field also allows wildcard domains like `*.example.com`, which would be valid for any subdomain of `example.com`, but not the top-level domain itself. Thus, wildcard names are often paired with another entry for the top-level domain to allow coverage of the whole domain. The vast majority of certificates have a relatively small number of items in their SAN fields (Figure 7). Between counts of 5 and 100, odd counts are orders of magnitude more common than even counts. This is because large SSL providers often bundle together a number of domains and wildcards for many different clients into a single certificate. Such large SAN counts are potentially dangerous, yet over 13 million certificates have a SAN count above 8. Though there is no limit on the number of items contained within the SAN field, there is a significant drop off with counts above 100. This may be due to limits in popular software or browsers, though this warrants further investigation to be certain.

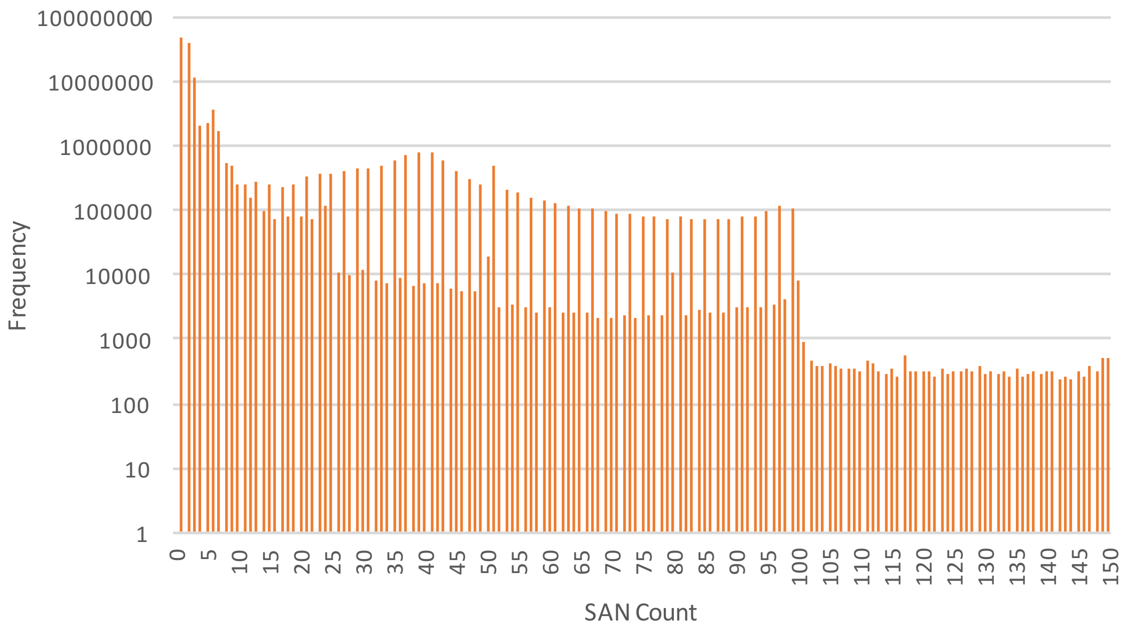

Figure 7: A histogram of SAN counts, with a logarithmic y-axis.

4.2 Suspicious Certificates

A number of certificates were marked as suspicious for a variety of reasons, ranging from the circumstances under which it was published to its validity in operating system trust stores.

4.2.1 Tunisian Root Certificate

Because the MD5 hashing algorithm has been considered insecure for a number of years, the logs were searched for certificates that use MD5 and are still valid today. One such certificate was the root certificate for Tunisia's national CA[2]. Its certificate revocation list (CRL) location returned a *404 Not Found* error, and yet the certificate's expiry date was set to the year 2037. This certificate, until a few years ago, was in the Microsoft Trust stores as a root certificate, which means that Microsoft software and browsers would have automatically trusted this certificate and anything that was signed using it. Though it has since been removed, it is likely that a portion of computers still trust this certificate, especially given

[2]`https://crt.sh/?id=18579`

that its CRL returns a server error. In addition, this certificate was created for the Tunisian CA for use on country domains, and therefore it is recommended that this certificate be restricted to `.tn` domains. Since this is not the case, this already weak certificate could be used to issue certificates and conduct man-in-the-middle (MITM) attacks on any domain.

4.2.2 SHA1 Certificates for 2018

Like the MD5 hashing algorithm, SHA1 is also considered insecure. The CA/B Forum, the overlying "regulatory" organization of CAs and browsers, resolved that CAs must not issue any subscriber certificates using SHA1 beginning in 2016 [27], and that "CAs must not issue OCSP responder certificates using SHA-1" beginning in 2017 [20, sec. 1.2.2]. Despite these requirements, eight certificates were found using SHA1; all eight of them had `not_before` dates in 2018 (in the future) and contained the phrase "OCSP" within the subject field[3]. These certificates are not explicitly in violation of the CA/B Forum Baseline Requirements, as they were issued in December 2016, before the rules regarding SHA1 in OCSP responders went into effect. However, it is known that SHA1 should not be used in certificates, and the existence of these certificates violates the intent of the aforementioned clauses.

GlobalSign, the CA to which all these certificates chain, maintains that these certificates exist solely for backward-compatibility purposes [28]. The pre-generation of OCSP signing certificates is an unfortunate and dangerous consequence of legacy systems not updating to support the latest, more secure cryptographic algorithms. It also raises questions about the language used within the CA/B Forum Requirements: as the requirement is clearly intended to curtail the usage of SHA1 in digital certificates, then the ease with which it can be circumvented undermines the rule as a whole.

It should be noted that browser vendors do take the Baseline Requirements quite seriously. For example, the certificate authorities WoSign and StartCom were distrusted by

[3]`https://crt.sh/?id=62407589`; `https://crt.sh/?id=62416636`; `https://crt.sh/?id=62423790`; `https://crt.sh/?id=62423799`; `https://crt.sh/?id=62423818`; `https://crt.sh/?id=62423833`; `https://crt.sh/?id=62423686`; `https://crt.sh/?id=62423690`

Apple, Google, and Mozilla for backdating certificates to circumvent the CA/B Forum's SHA1 deprecation requirements [29].

4.2.3 Certificates Using 512-bit RSA

In addition to examining certificates using weak hash algorithms, the logs were scanned for certificates using small RSA key sizes. Benjamin Moody factored a 512-bit RSA key on a single computer in 2009. In 2015, a group was able to factor a 512-bit RSA key for a low of $75 in Amazon Web Services' EC2 cloud [30]. Based on these vulnerabilities, 512-bit RSA keys have been deprecated by the CA/B Forum. In fact, 1024-bit keys have also been deprecated since 2014.

There are 2070 certificates in the log with a key size of 512-bits, of which 35 certificates have `not_after` dates in 2014 or later. Five of these certificates should be using 2048-bit keys according to the Baseline Requirements, and the other 30 should be using 1024-bit keys. Two certificates are particularly egregious: one was issued in 2015 and expires in 2017[4], the other was issued in 2016 and valid until 2018[5]. In addition, the second certificate is for `kaba.tech`, a domain that has since expired and thus is available for registration.

The private key for this certificate was obtained using a factorization attack. This utilized under 42 hours of computing time on a single 64-core, `m4.x16large` AWS instance. The factorization itself was done using the CADO-NFS, the same software that was used to factor RSA-704 and RSA-768 [31]. Because the `kaba.tech` was unregistered, it would have been possible to register that domain and operate it on behalf of original certificate's subject, completely without their knowledge. It would enable TLS-secured communications and certificate verification without having ever requested a certificate. However, this certificate had been issued by WoSign, a CA that had already been blacklisted or removed from most major trust stores, and therefore was not accepted. Some browsers had additional checks that would reject the certificate on the basis of its small key size, but not all browsers have

[4]`https://crt.sh/?id=6983186`
[5]`https://crt.sh/?id=31030664`

implemented such a feature.

The mere existence of 512-bit key sizes for RSA is alarming because of how few resources it takes to crack them. Cracking such a certificate would allow an attacker to impersonate the destination server or conduct man-in-the-middle attacks. The current proposal of HTTP Public Key Pinning (HPKP), which intends to reduce the impact of compromised CAs, would also be inefficacious in detecting a cracked public key [5]. These certificates using 512-bit RSA offer limited security and should be revoked and reissued with stronger cryptography.

4.2.4 Badly Encoded Certificate

Given the list of certificates that the Python script was unable to parse properly, these errors were cross-referenced with *certlint* and OpenSSL, both of which also parse and report errors within certificates.

There was one certificate that caused issues with all three programs[6]. This particular certificate had a number of errors. It contained a negative serial number, which is not allowed in X.509 certificates. Moreover, the serial number integer was improperly encoded in terms of padding, which caused certain versions of OpenSSL to crash. The older version of OpenSSL, `0.9.8zh`, was able to parse and report the certificate's errors properly, but versions `1.0.2l` and `1.0.10f` report a decoding error and exit with a non-zero exit code when given this certificate. While this is not a bug in OpenSSL, as the reduced leniency was introduced by design, it is worrying that such a certificate was ever created and signed by a CA. It also raises questions about how this certificate was accepted into a certificate log to begin with.

5 Conclusion

The project presented harnesses the expansiveness of an existing Certificate Transparency log to conduct a more comprehensive survey of the SSL ecosystem. This survey revealed promising trends in certificate lifetimes and signature algorithm usage and quantified the

[6]`https://crt.sh/?id=1914312`

ubiquity of potentially dangerous large SAN counts. In addition, it examined a number of individual certificates, revealing some dangerous practices. It revealed a wildly unsecured root certificate from Tunisia with both the potential to be compromised and an attack vector in outdated Microsoft Windows-based devices. It also uncovered eight certificates using SHA1 with validity periods in 2018, a circumvention of CA/B Forum Baseline Requirements with intent to use a deprecated and insecure hashing algorithm. Thirty-five recent certificates still use 512-bit keys, of which one certificate has not yet expired and could be used in potential attacks. In addition, one certificate was found to cause certain versions of OpenSSL to exit with an error while decoding because of a bad certificate encoding. A number of recommendations were also presented in relation to the issues described. While not nearly enough for a full certificate monitor, this work highlights characteristics of certificates that monitors should examine when looking for suspicious certificates.

6 Acknowledgments

I would like to acknowledge my mentors, Samuel Erb and Tom Houman of Akamai Technologies, for their patience and guidance throughout. I would also like to thank Jenny Sendova, Andrew Wang, Ersin Arioglu, and Jiwoo Lee for reviewing my many drafts. In addition, I am gracious for my sponsors' support. Finally, I would like to thank the Massachusetts Institute of Technology, the Center for Excellence in Education, and the Research Science Institute for making this research possible.

References

[1] Google. Transparency Report. Available at https://www.google.com/transparencyreport/https/ (2017-07-21).

[2] Let's Encrypt Stats. Available at https://letsencrypt.org/stats/ (2017-08-01).

[3] S. Shankland. Google enlists Chrome in push for encrypted Web. Available at https://www.cnet.com/news/chrome-becoming-tool-in-googles-push-for-encrypted-web/ (2017-07-08), Jan. 2015.

[4] T. Scott. HTTPS-everywhere for government. Available at https://obamawhitehouse.archives.gov/blog/2015/06/08/https-everywhere-government (2017-08-01), June 2015.

[5] T. Scott. A Policy to Require Secure Connections across Federal Websites and Web Services. White House Office of Management and Budget. Available at https://www.whitehouse.gov/sites/whitehouse.gov/files/omb/memoranda/2015/m-15-13.pdf, 2015.

[6] I. Ristic. Internet SSL survey. *Talk at Black Hat Abu Dhabi*, 2010.

[7] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 291–304, New York, NY, USA, 2013. ACM. Available at http://conferences.sigcomm.org/imc/2013/papers/imc257-durumericAemb.pdf.

[8] J. K. Harris. Understanding SSL/TLS. Available at https://computing.ece.vt.edu/~jkh/Understanding_SSL_TLS.pdf (2017-07-20), Oct. 2008.

[9] 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, May 2008.

[10] Yanpan (Own work). Chain of trust. https://upload.wikimedia.org/wikipedia/commons/d/d1/Chain_of_trust.svg (distributed under CC BY-SA 4.0, via Wikimedia Commons), 2016.

[11] S. B. Roosa and S. Schultze. The "Certificate Authority" Trust Model for SSL: a Defective Foundation for Encrypted Web Traffic and a Legal Quagmire. *Intellectual Property & Technology Law Journal*, 22(11):3, Nov. 2010. Available at https://citpsite.s3.amazonaws.com/publications/Roosa_Schultze_CA_Trust_Model.pdf (2017-07-21).

[12] Certificate Transparency Team. What is Certificate Transparency? Available at https://www.certificate-transparency.org/what-is-ct (2017-07-21).

[13] B. Laurie. Certificate Transparency. *Queue*, 12(8):10:10–10:19, Aug. 2014.

[14] B. Morton. Certificate Transparency deadline moved to April 2018. *Entrust Blog*, May 2017. Available at `https://www.entrust.com/certificate-transparency-deadline-moved-april-2018/` (2017-08-01).

[15] Certificate Transparency Team. Known logs. Available at `https://www.certificate-transparency.org/known-logs` (2017-07-20).

[16] SSLMate. Certificate Transparency log growth. Available at `https://sslmate.com/labs/ct_growth/` (2017-07-21).

[17] T. Ritter. Certificate Transparency tools. GitHub repository available at `https://github.com/tomrittervg/ct-tools`, 2017.

[18] Python Cryptographic Authority. Cryptography. GitHub repository available at `https://github.com/pyca/cryptography/`, 2017.

[19] Amazon Web Services Labs. X.509 Certificate Linter. GitHub repository available at `https://github.com/awslabs/certlint`, 2017.

[20] CA/Browser Forum. Baseline requirements certificate policy for the issuance and management of publicly-trusted certificates. Available at `https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.4.9.pdf` (2017-07-29), 2017. Version 1.4.9.

[21] OpenSSL Software Foundation. OpenSSL Project. GitHub repository available at `https://github.com/openssl/openssl`. Online at `https://www.openssl.org/` (2017-07-21).

[22] COMODO CA Limited. Certificate search. Available at `https://crt.sh/`, 2017.

[23] J. Aas. Milestone: 100 Million Certificates Issued. *Let's Encrypt*, June 2017. Available at `https://letsencrypt.org/2017/06/28/hundred-million-certs.html` (2017-07-08).

[24] M. Stevens, A. Sotirov, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. De Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *Crypto*, volume 5677, pages 55–69. Springer, 2009.

[25] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The first collision for full SHA-1. *IACR Cryptology ePrint Archive*, 2017:190, 2017.

[26] A. K. Lenstra, X. Wang, and B. de Weger. Colliding X.509 certificates, 2005.

[27] CA/B Forum. Ballot 118 SHA-1 sunset (passed). Available at `https://cabforum.org/2014/10/16/ballot-118-sha-1-sunset/` (2017-07-21), Oct. 2014.

[28] D. Beattie. SHA-1 usage in OCSP responder. Mozilla's dev-security-policy Mailing List, Aug. 2017. Available at `https://groups.google.com/forum/#!topic/mozilla.dev.security.policy/aCJQ5JkYcVw`.

[29] A. Whalley. Distrusting WoSign and StartCom certificates. Available at https://security.googleblog.com/2016/10/distrusting-wosign-and-startcom.html (2017-08-01), Oct. 2016.

[30] L. Valenta, S. Cohney, A. Liao, J. Fried, S. Bodduluri, and N. Heninger. Factoring as a service. Cryptology ePrint Archive, Report 2015/1000, 2015. http://eprint.iacr.org/2015/1000.

[31] The CADO-NFS Development Team. CADO-NFS, An Implementation of the Number Field Sieve Algorithm. http://cado-nfs.gforge.inria.fr/, 2015. Release 2.3.0.

[32] M. Stevens, A. K. Lenstra, and B. De Weger. Chosen-prefix collisions for MD5 and applications. *International Journal of Applied Cryptography*, 2(4):322–359, 2012.

[33] D. Song. Hash Functions, MACs, Digital Signatures. Available at https://inst.eecs.berkeley.edu/~cs161/fa08/Notes/lec4-hash-mac-sig.pdf (2017-07-21), 2008.

[34] Q. H. Dang. Secure hash standard. *Federal Inf. Process. Stds.(NIST FIPS)-180-4*, 2015.

[35] H. Sheth. Kaba.tech proof of private key possession. GitHub repository available at https://github.com/hsheth2/kaba.tech-certificate-proof, 2017.

Appendix A Certificate Digital Signatures

To prevent a certificate from being modified and to verify the issuer's identity, its contents are digitally signed by the issuer. First, the contents of the certificate are hashed using a cryptographically secure hash function. Hash functions are deterministic, one-way functions for which it is difficult to find two inputs that give the same hash output. More formally, cryptographic hash functions should have the following properties [32]:

- Collision resistance: It should be difficult to find an m_1 and m_2 for which $\text{hash}(m_1) = \text{hash}(m_2)$.
- Second pre-image resistance: It should be difficult to find m_2 for a given m_1 such that $\text{hash}(m_1) = \text{hash}(m_2)$. Collision resistance implies second pre-image resistance [33].
- Pre-image resistance: It should be difficult to find m such that $\text{hash}(m) = h$ for a given h. This is related to the idea of one-way functions.

Popular cryptographic hash functions include the SHA2 and SHA3 families; the older MD5 and SHA1 hash algorithms are not considered secure [34]. Collisions with MD5 were found in 2005 [26], and in 2009 were used to create rogue certificates [24]. Hash collisions have also been found for SHA1 [25]. These secure hash functions are used to irreversibly compress the certificate into a specified-length hash. Next, the issuer performs a set of operations on this hash using its own private key and embeds the resulting signature within the certificate. The digital signature process can only be performed by the issuer of the certificate, as that entity is the sole owner of its private key. However, the digital signature can be verified using only the issuer's public key, which is public information. In this way, anyone can verify a certificate's signature, but only the issuer can create that signature.

Appendix B `kaba.tech` Private Key

The public key was contained within the certificate at `https://crt.sh/?id=31030664`. Although the private key was recovered, it would not be prudent to release it. However, proof that it was recovered is given in the form of a digital signature. The file that was signed contained the following text (without quotes): "This file was digitally signed on 2017-08-01.". Its base64 encoded signature using SHA256 is `FOLvWOIB5Ps53jq03MTf6FKqieqHRMbJt467OPTX zOSNa7D1TLsZlW3eo/CsTD2JIIZMT6bfsnWA6S3PeiSWGg==`. This signature can be verified using the following command:

```
$ openssl dgst -sha256 -verify pubkey.pem \
    -signature base64_decoded_sig original_file
```

Alternatively, this proof is contained within a GitHub repository, which contains all of the necessary files and scripts for the verification command [35].