ABSTRACT

Title of dissertation: UNDERSTANDING ADVERSARY BEHAVIOR AND SECURITY THREATS IN PUBLIC KEY INFRASTRUCTURES

Doowon Kim
Doctor of Philosophy, 2020

Dissertation directed by: Professor Tudor Dumitraș
Department of Computer Science

Public Key Infrastructure (PKI) is designed to guarantee the authenticity and integrity of digital assets such as messages, executable binaries, etc. In PKIs, there are two representative applications: 1) the Web PKI and 2) the Code-Signing PKI. 1) The Web PKI enables entities (e.g., clients and web service providers) to securely communicate over untrusted networks such as the Internet, and 2) the Code-Signing PKI helps protect clients from executing files of unknown origin. However, anecdotal evidence has indicated that adversaries compromised and abused the PKIs, which poses security threats to entities. For example, CAs have mis-issued digital certificates to adversaries due to their failed vetting processes. Moreover, private keys that are supposed to be securely kept were stolen by adversaries. Such mis-issued certificates or stolen private keys were used to launch impersonation attacks. In this regard, we need to have a sound understanding of such security threats and adversaries' behaviors in the PKIs to mitigate them and further to enhance the security of the PKIs.

In this dissertation, we conduct a large-scale measurement study in the two representative applications—the Web PKI and the Code-Signing PKI—to better understand adversaries' behaviors and the potential security threats. First, in 1) the Web PKI, we mainly focus on phishing websites served with TLS certificates. From the measurement study, we observe that certificate authorities (CAs) often fail in their vetting process and mis-issue TLS certificates to adversaries (i.e., phishing attackers). Also, CAs rarely revoke their issued TLS certificates that have been compromised. Second, in 2) the Code-Signing PKI, we characterize the weaknesses of the three actors (i.e., CAs, software publishers, and clients) that adversaries can exploit to compromise the Code-Signing PKI. Moreover, we measure the effectiveness of the primary defense, *revocation*, against the Code-Signing PKI abuses. We find that erroneous revocations (e.g., wrong effective revocation date setting) can pose additional security threats to clients who execute binaries because the revocations become ineffective. Such security threats stem from an inherent challenge of setting an effective revocation date in the Code-Signing PKI and CAs' misunderstanding of the PKI. These findings help Anti-Virus companies and a CA fix their flaws.

UNDERSTANDING OF ADVERSARY BEHAVIOR AND SECURITY THREATS IN PUBLIC KEY INFRASTRUCTURES

by

Doowon Kim

Submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Tudor Dumitraș, Chair/Thesis Advisor
Professor Ashok Agrawala, Dean's Representative
Professor Nirupam Roy
Professor Dana Dachman-Soled
Doctor Chris Gates

Acknowledgments

With much gratitude, to my beloved wife, Kyungjin, and my families in Suwon, Jeju, and Virginia who endlessly helped me take small steps on a long journey toward being a researcher, and my thesis advisor, Dr. Tudor Dumitraș who nurtured and fostered me along the way.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Securely communicating with other entities and exchanging digital assets over untrusted networks such as the Internet is significantly challenging. Adversaries can impersonate other people, eavesdrop digital communications, and alter digital assets for their malicious purpose. For example, an adversary is able to conduct a man-in-the-middle attack (MITM) against an online banking service. In this attack, an adversary is placed in the middle of two entities (i.e., client and online bank) and she/he can eavesdrop and steal the client's ID and password for her/his online banking service.

To this end, *Public Key Cryptography* was designed and introduced. In the cryptographic technique, to guarantee the authenticity and integrity of digital assets, a pair of two keys, a *public key* and a *private key* are required. A private key must be securely kept by an owner and the key is used to sign digital assets such as digital messages and executable binaries, called a digital signature. In turn, another key, a public key is publicly available to other people including even adversaries. The key is used by anyone to verify a digital signature. A successful verification with the public key indicates that the digital assets are never altered after signed (i.e., integrity) and they are originated from the owner of the private key (i.e.,

authenticity). Furthermore, even though a public key is associated with a private key is publicly available, it is computationally infeasible for anyone including adversaries to derive the private key from the public key because they are mathematically related and guaranteed not to readily break.

Another challenge inherent to Public Key Cryptography is the difficulty of establishing trust in the ownership of public keys. In other words, we are unable to know whether public keys are legitimately originated from its real owner of the associated private keys. Some adversaries can impersonate someone else and send their malicious public keys to victims. In turn, victims are unable to derive the owner's information from the public key because the public key has no information regarding the owner of the associated private key. To this end, a *Public Key Infrastructure* (PKI) is designed. In a PKI, there are trusted third-parties, called *Certificate Authorities (CAs)* who are supposed to verify the owner of a public key and bind the public key to the owner's information. The binding information—including the owner's information and the owner's public key—are specified in an X.509 v3 certificate [5], called a digital certificate. Entities who want to securely communicate use this digital certificate to verify the public key and the ownership of the public key.

In PKIs, the two representative applications are 1) **the Web PKI** and 2) **the Code-signing PKI**. 1) The Web PKI is designed and deployed for secure communication on the Internet. In the PKI, it helps entities (i.e., clients) authenticate web servers and guarantee the integrity of messages between clients and web servers. As of April 2020, more than 90% Internet traffic is securely encrypted, which means

that most websites are served in the Web PKI [6]. Another PKI, 2) the Code-Signing PKI is used for executable binaries. In the PKI, entities (i.e., clients) are able to establish trust in signed binaries. In other words, they can know who has published signed binaries and whether or not signed binaries are altered.

However, PKIs have been compromised by adversaries for their malicious purposes such as economic or political gains [7–14]. For example, in the Web PKI, adversaries exploited the *DigiNotar*'s (a CA) issuance systems and fraudulently issued a wildcard certificate for Google. Using the misissued Google certificate, the adversaries conducted a man-in-the-middle (MITM) attack against Google services in Iran [7]. Furthermore, in the Code-Signing PKI, other adversaries stole private keys from reputable two Taiwanese companies and signed their malware, *Stuxnet* with the stolen private keys [14]. In this regard, we need to have a deep understanding of potential security threats and adversaries behaviors in the PKIs to mitigate them and further to enhance the security of the PKIs.

In this dissertation, we conduct a large-scale measurement study in the two representative PKIs—1) the Web PKI and 2) the Code-Signing PKI—to better understand security threats and adversary behaviors. This dissertation consists of a series of measurement studies in the Web PKI (Chapter 4) and the Code-Signing PKI (Chapter 5). In these measurement studies, we aim to understand adversary behaviors—such as how they compromise the two PKIs for their malicious goals—and identify new potential security threats.

The Web PKI. This dissertation begins with understanding the current landscape

of the security threats in the Web PKI (Chapter 4). We mainly focus on phishing attacks in the Web PKI. Specifically, we attempt to better understand how adversaries abuse the Web PKI for their phishing attacks.

Modern web browsers such as Mozilla Firefox, Google Chrome, etc. display a green or gray padlock icon in its URL bar when clients access web services served in the Web PKI (i.e., TLS). Technically, the green padlock icon indicates that the authenticity and integrity of a website that a client communicates are guaranteed. However, unfortunately, many clients (potential victims) incorrectly believe that the green padlock icon in the modern browsers means that phishing attacks can be prevented (Section 4.1) [15–18].

Adversaries can take advantage of this misunderstanding of the green padlock icon by serving their phishing websites with valid TLS certificates. Typical users may be tricked by phishing websites deployed with valid TLS certificates because valid TLS certificates can be presented as a green padlock icon in the users' web browsers. Moreover, according to the report of Anti-Phishing Working Group (APWG), approximately more than 60% of phishing websites are launched with TLS certificates in the first quarter of 2019 [19]. This indicates that more TLS phishing attacks occur than plain HTTP phishing attacks do. In this regard, we need to understand how adversaries abuse the Web PKI (i.e., TLS) for their phishing attacks.

We first attempt to understand the current landscape of the phishing attacks with TLS certificates (the Web PKI). We analyze 5.05M phishing URLs collect from *eCrimeX*, and find that 1) CAs rarely revoke abusive certificates and 2) a CA's vetting process often fails (as described in Section 4.2).

The Code-Signing PKI. In the second part of the dissertation, we mainly focus the Code-Signing PKI; specifically, 1) we fully characterize the weaknesses of the three actors (certificate authorities, software publishers, and end-users) in the PKI that adversaries can exploit (Section 5.3), 2) we systemically measure the security threats of the Code-Signing PKI in the wild, 3) we also aim to understand the underground economy of code-signing certificate (Section 5.4), and 4) we measured the effectiveness of the primary defense, *revocation*, against the abuse (Section 5.5).

Adversaries always aim to generate sophisticated malware for economic or political gain. More importantly, they also seek to effectively distribute their malware to end-users (victims) and to lure the end-users to install the malware while concealing its identity. The sophisticated malware, *Stuxnet* can be a representative example. The malware was signed with the stolen private keys of two code-signing certificates from reputable Taiwanese companies, which helped malware authors hide their identities and lured end-users to readily establish trust in the signed malware. Even worse, the signed malware remained undetected for a longer period because Anti-virus companies neglected to perform further inspection as they were properly signed with code-signing certificates of the well-known companies [14].

As observed, adversaries have abused the Code-Signing Public Key Infrastructure (PKI) for their malicious goals, and their exploits were significantly effective. However, little attention has been paid on such abuses in the Code-Signing PKI because security researchers presumably believed that the abuse was uncommon in the wild and the security threats were the same as in the Web PKI (e.g., TLS). Moreover, the Code-Signing PKI inherently has a challenge of Code-Signing certificate

collection in the wild due to the nature of software distribution while in the Web PKI, TLS certificates can be readily collected via network scanners (e.g., ZMap).

We identify the root causes of the abuses in the Code-Signing PKI. We began by fully characterizing the weaknesses of the three actors (certificate authorities, software publishers, and end-users) in the PKI that adversaries can exploit (Section 5.3). We then systemically measured the security threats of the Code-Signing PKI in the wild to understand the prevalence of malware signed with abusive certificates after overcoming the challenge of certificate collection (Section 5.3 and Section 5.4). We also aim to understand the underground economy of code-signing certificates (Section 5.4). Last, we measure the effectiveness of the primary defense, *revocation*, against the abuses (Section 5.5). We highlight the ineffective revocation can lead end-users to remain exposed to the security threats although such abusive certificates have been already revoked by the issuing CAs. Our findings helped Anti-Virus companies and a certificate authority (CA) fix their flaws stemmed from a misunderstanding of the Code-Signing PKI. Our results and datasets are available at *http://signedmalware.org*.

1.1 Published Works and Copyrights

This dissertation extends the materials from four papers by the author Doowon Kim [1–4]; three of them have been already published at ACM CCS 2017, WEIS 2018, and USENIX Security 2018 and another paper is in submission.

Chapter 2: Background Principles

In this chapter, I overview Public Key Infrastructures (PKIs): particularly, two representative PKIs, 1) the Web PKI (i.e., TLS) and 2) the Code-Signing PKI (i.e., Windows Authenticode).

2.1 Overview of Public Key Infrastructure (PKI)

A Public Key Infrastructure (PKI) relies on the digital signature mechanism to enable entities (e.g., clients and service providers) to securely exchange the digital assets (e.g., messages, email, software, etc.) over untrusted networks such as the Internet. The digital signature mechanism guarantees the authenticity and integrity of digital assets, and requires a signer (i.e., a service provider) to generate a pair of public and private keys using a mathematical algorithm such as RSA. A private key is always securely kept by the signer and is used to sign a digital asset; another pair key, public key, associated with the private key is publicly available and is used to verify the digital signature. If the digital asset is altered after signed, the signature is no longer verified and trusted.

However, the main problem of the digital signature mechanism is that in the untrusted networks, how we can establish trust in public keys legitimately origi-

nated from the owner of the private keys. In other words, adversaries can impersonate someone else and send their malicious public keys to victims. In PKIs, there are trusted entities, called Certificate Authorities (CAs) in whom all entities (e.g., service providers and clients) can establish trust. When service providers (e.g., websites or software publishers) want to provide clients with signed digital assets and the associated public keys, they need to apply for digital certificates issued from Certificate Authorities (CAs). After carefully verifying the applicants' identities, the CAs issue digital certificates that bind the applicants' public keys to their identities. In order to prevent the digital certificates from being forged, the leaf certificates are also signed with the issuing CA's private key of the intermediate certificate and then signed with the private key of the root certificate. When a client verifies a digital signature created for a digital asset, the client has to validate each digital signature using a *chain of trust*; the trust anchor is the root certificate pre-installed on client's machine such as operating systems, web browsers, etc. Moreover, they are also responsible to revoke compromised digital certificates that they have issued after they discover that their issued certificates are misused for malicious purposes.

2.2 The Web PKI (TLS)

In PKIs, one of the most representative examples is the Web PKI (e.g., Transport Layer Security). Secure Sockets Layer (SSL) and Transport Layer Security (TLS) aim to provide privacy and data integrity between two entities, servers and clients. In other words, the protocols have been designed to prevent eavesdropping

and tampering such as a Man-in-the-middle attack. Therefore, TLS/SSL becomes the de facto standard for the secure communications over the untrusted networks such as the Internet. More than 90% of the Internet traffic was encrypted and securely communicated over TLS/SSL in 2020 [20]. The TLS/SSL protocols rely on a PKI, called the Web PKI. In the ecosystem, Certificates Authorities (CAs) are a publicly trusted third-party entity. They are able to issue digital certificate (i.e., TLS certificates) after vetting processes. Moreover, another responsibility is *revocation*. In this process, they revoke their issued TLS certificates when they discovered that the certificates are mis-issued or misused for malicious purposes.

2.2.1 Type of CAs

Initially, there had been only a few commercial CAs (e.g., Comodo, DigiCert, GoDaddy, etc.). Website owners or administrators had to pay for TLS certificates to obtain certificates and deploy their websites with the issued certificates. However, recently, a new method, called Automated Certificates Management Environment (ACME) protocol has been designed and deployed [21], which leads to the wide adoption of TLS. In this protocol, human-involvements such as applicants' verification process are reduced as much as possible. Therefore, the ACME CAs do not charge any price to applicants (i.e., website administrators) when issuing TLS certificates. Due to this price reason, in April 2020, more than 90% of the Internet traffic have been securely communicated over TLS [20].

2.2.2 Required Practices for CAs

There are standard guidelines where basic requirements of CAs are manifested: 1) CA/Browser Baseline Requirements [22] and 2) Certificate Practice Statement (CPS). CA/Browser Baseline Requirements manifests a standard guideline that CAs should follow. In addition, each CA publishes its own practice statements, called Certificate Practice Statement (CPS). Each CPS is specified at its x.509 v3 certificate [23].

2.2.2.1 CA/Browser Baseline Requirements

The voluntary business association of CAs, web browsers, and other PKI-related groups, called Certification Authority Browser Forum (CA/Browser Forum), has publicized the guidelines of the issuance and management of X.509 digital certificates (e.g., revocation and how to securely manage private keys). Baseline Requirements [22] is the current standard guideline, and was published in 2011 and became effective from June 2012.[1] The standard guideline covers all type of TLS certificates such as DV (Domain Validation), OV (Organization Validation), and EV (Extended Validation). Furthermore, all CAs should comply with Baseline Requirements.

2.2.2.2 Certificate Practice Statement (CPS)

In addition to CA/B BRs (Baseline Requirement), each CA specifies its own rules of practices for management procedures; it is called Certificate Practice State-

[1]The current version (1.6.5) was published on April 16, 2019.

ment (CPS). Particularly, such the document specifies how to verify applicants, how to issue TLS certificates, and how/when they revoke certificates if compromised or misused. CPS must be specified in the Certificate Policies extension of an X.509 v3 certificate [23].

2.2.3 Type of TLS Certificates

Typically, there are three types of TLS certificates depending on CAs' validation procedures. First, Domain Validated (DV) certificates are the most commonly used in the wild because the ACME protocol was designed and deployed, and the protocol supports such certificates at free charge. The DV certificates require the most minimal validation process where CAs only check whether applicants own domains. In other words, they see only the ownership of domains to which certificates will be issued. The ACME CAs provide free TLS certificates for applicants (i.e., webserver administrators), which leads to the wide adoption of TLS certificates in the wild.

Second, Organization Validated (OV) certificates are issued only after the issuing CAs check if domains are owned by the applicants and the applicants' information is legitimate. web browsers tend to display the applicants' information (i.e., website companies). The price of OV certificates is higher than DV certificates because CAs should be involved in the issuance procedure when they check the applicants' identity such as government-issued ID or business information.

Last, Extended Validation (EV) certificates are the highest-ranking and most

expensive TLS certificates. Applicants who want EV certificates need to go through the extended identity verification process to assert that they are authorized and exclusively own the rights to use the domains. In turn, web browsers are likely to display information of the domain owner such as company name, which helps clients ensure that they access legitimate websites.

2.2.4 Revocation in the Web PKI

CAs have another important responsibility besides the issuance of TLS certificates, which is *revocation*. Revocation means that the issuing CAs make their issued certificates no longer valid when they become compromised and misused. CAs can revoke TLS certificates for a variety of reasons; (1) certificates are fraudulently issued to adversaries (e.g, DigiNotar) [7,24,25], (2) the associated private keys became compromised (e.g., Heartbleed) [26], and (3) the issued certificates have been misused (e.g., phishing attacks). Moreover, when the CAs are made aware that their issued TLS certificates no longer comply with the CA/Browser Forum Baseline Requirements [22] such as a cryptographically weak key used for TLS certificates [27], they also should revoke the certificates.

After they revoke TLS certificates, they need to disseminate the revocation information to clients. Currently, two dissemination methods are being widely used: (1) Certificate Revocation List (CRL) and (2) Online Certificate Status Protocol (OCSP) [5, 28].

2.2.4.1 Certificate Revocation List (CRL)

A CA publishes CRL files that include revoked certificates information such as serial numbers, revocation dates, and revocation reasons [5]. CAs have to update and reissue their CRL files every a certain period; particularly, according to the Baseline Requirements, they should do at least once every seven days. The locations (URLs) of the CRLs files are specified at the CRL Distribution Point (CDP) of X.509 v3 certificates. In turn, clients should periodically access/download the CRL files and then check if a TLS certificate is listed in the CRLs. The main problem of this dissemination method is that clients have to download all unnecessary revoked certificate information in which clients are not interested. The size of a single CRL file has increased to even 76MB [29], which is significantly cumbersome for clients.

2.2.4.2 Online Certificate Status Protocol (OCSP)

This protocol was introduced to overcome the challenge of the network overheads in the CRLs. In this protocol, clients are able to simply query an OCSP server for a TLS certificate that they want to check the revocation status [28]. This helps reduce the network overheads on the client- and server-side as well. The OCSP point is specified at Authority Information Access (AIA) of X.509 v3 certificates.

2.2.4.3 OCSP Stapling

Even though OCSP reduces the huge network overheads of CRLs, OCSP still has a network latency because clients are supposed to query an OCSP server before

establishing the trust in a TLS certificate of a certain website that the clients access. In other words, the clients wait and cannot access the websites until OCSP responses are sent to the clients. To reduce this latency, OCSP Stapling was introduced [30]. In this protocol, a website pre-fetches the revocation status of its TLS certificate and sends clients the revocation status embedded in its TLS handshake when the clients attempt to connect the website. This reduces the latency of the original OCSP.

2.2.5 Certificate Transparency (CT) Logs

A reseller of the CA, *Comodo*, was compromised in 2011 and mis-issued a wildcard certificate for Google. Adversaries in Iran to conduct man-in-the-middle attacks against Google using the mis-issued TLS certificate [8,9]. To mitigate these misissuance problems, a new system is designed and deployed in 2020, called *Certificate Transparency (CT)* [31].

CT depends on publicly-available, verifiable certificate transparency logs. The logs can be only appended using *Markle hash tree* so that no one is able to modify the logs. In other words, when TLS certificates are issued, immediately the issued certificates are logged in CT logs, but the already-logged TLS certificates are unable to be modified or removed. Third-parties are able to audit and monitor issued TLS certificates using CT. They are readily able to detect any misissuance of TLS certificates from CT logs. Additionally, CT becomes a de-facto data source for TLS certificate research because it logs almost-all TLS certificates in the wild.

2.3 The Code-Signing PKI

Another application in Public Key Infrastructures (PKIs) is the Code-Signing PKI [32]. The PKI also utilizes the digital signature mechanism to guarantee the authenticity and integrity of executable binaries. In other words, if an executable binary is properly signed by a software publisher, clients are able to identify the software publisher (i.e., authenticity), and check whether or not the signed software program is altered after the software program is signed (i.e., integrity).

The CAs whom clients and software publishers can establish trust can issue Code-Signing certificates to software publishers. Software publishers can use their private keys associated with the issued Code-Signing certificates to sign their software programs. In turn, since clients trust the issuing CAs, they can use the public keys embedded in the certificates issued by the CAs to verify the digital signatures bundled with the signed software programs.

Code-Signing certificates issued from CAs can be used to sign as numerous software programs as the owners of the certificates want; however, in TLS, a TLS certificate is typically bound to a certain domain since the domain name is specified in the common name field and clients validate the common name and the domain name when they try to access.

Various application platforms and major operating systems including Microsoft Windows, Apple macOS, iOS, Android, Linux, etc., support the code signing. It is widely used for Windows and macOS executables and drivers, Firefox XPIs, Android/iOS apps, Java Jars, Visual Basic for Applications, Adobe Air apps, etc.

Another feature of the Code-Signing PKI in Windows is the *trust updates* that are automatically assured and applied if the updates are signed with the same certificate used by its original program code.

The Windows platforms are dominant in terms of the computer market share, and prevalent malware in the wild is PE32 executables. Therefore, in this dissertation, I will mainly focus on PE32 executables and Windows Authenticode [32].

2.3.1 Windows Authenticode Code-Signing PKI

Windows Authenticode [32] is a code signing standard in the Windows platforms. It is used to sign Windows files such as Portable Executable (.exe), catalogs (.cat) files, ActiveX controls (.ctl, and .ocx), dynamically loaded libraries (.dll), cabinet files (.cab), etc. Authenticode also relies on Public Key Cryptography Standard (PKCS) #7 [33], and a PKCS #7-formatted content, called *signed data* that includes a chain of X.509 v3 [5] code signing certificates, a chain of x.509 v3 Time Stamping Authority (TSA) certificates, a digital signature, a signing date (trusted timestamp), etc. Authenticode supports MD5, SHA-1, and SHA-256 hashes; but currently MD5 and SHA-1 are no longer supported in Windows 10.

2.3.1.1 Protections that Rely on Code Signing

Microsoft SmartScreen and User Account Control (UAC). To combat malware, *Microsoft Defender SmartScreen* [34] have been designed to work with the Code-Signing PKI and (usually internal) databases of applications and publishers'

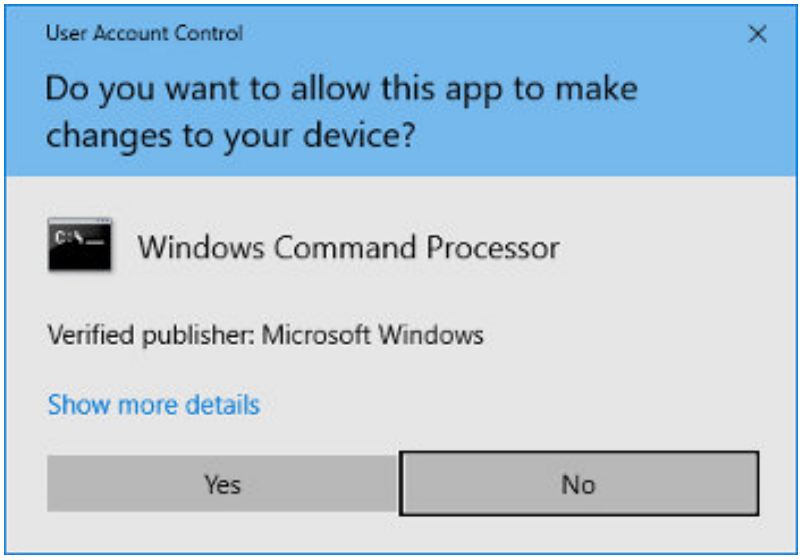

Figure 2.1: **Example of UAC Warning Dialog.**

reputation to assess if an application is safe to launch. *UAC* verifies a code signing signature and displays the publisher's name of the signature to clients when the program of the signature requests an elevated privilege. For example, when an executable is downloaded on a Windows 10 machine, SmartScreen attempts to assess its reputation before it allows the client to launch it, and when clients execute the downloaded binary sample that requests an elevated privilege, *UAC* verifies the embedded signature and shows the software publisher name to the clients as shown in Figure 2.1.

There are two types of code signing certificates; 1) standard code signing certificate and 2) Extended Validation (EV) code signing certificate. A standard code signing certificate requires reputation before distributed and executed. In particular, the reputation of binary samples signed with standard certificates can

improve over time when the signed samples garner a track record of installations on multiple hosts without raising suspicions. Without building up a reputation, Windows UAC presents a warning dialog to clients with "Unknown Publisher." This represents a challenge for malware developers who aim to avoid suspicion and to remain stealthy. Even if they manage to produce a valid signature for their malware, they must also ensure that their Authenticode certificate has accumulated a sufficient reputation in SmartScreen to prevent the user warning.

Meanwhile, an Extended Validation (EV) certificate can be issued with a stricter vetting processing specified in the Guidelines for Extended Validation produced by the CA/Browser Forum[2]. Binary samples signed with EV certificates can receive instant reputation in *SmartScreen*; in other words, EV certificates receive a good reputation initially. From Windows 10 (version 1607), a stricter requirement is set: EV certificates are necessary for any new kernel-mode drivers [35].

Safe Browsing. Google *Safe Browsing* is another protection system similar to the *SmartScreen*. The white paper mentions that "Chrome trusts potentially dangerous file types that match URLs in the white list, and it also trusts files signed by a trusted authority [36]."

Anti-Virus. Anti-Virus engines also utilize code signing information. To reduce false positives, some AV engines use whitelisting based on code signing certificates. For example, Symantec mention in their whitelisting page: "To prevent false positive detections we strongly recommend that you digitally sign your software with a class

[2]https://cabforum.org

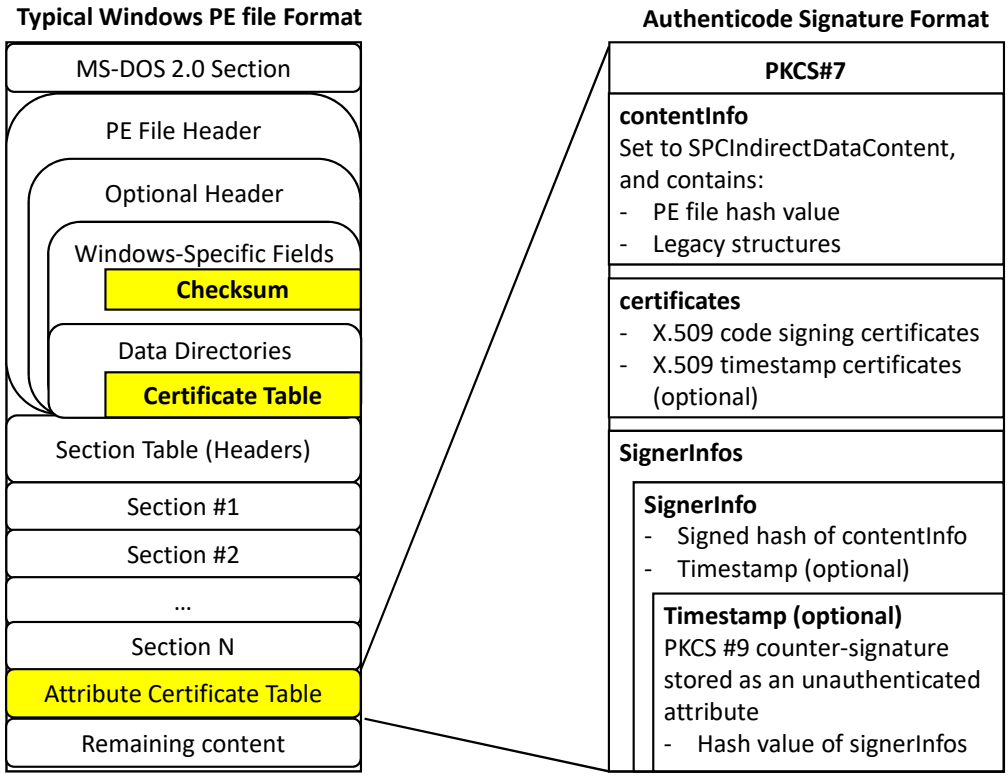

Figure 2.2: **Windows PE and Authenticode Signature Format.** The yellow background sections are excluded in the calculation of a code-signing signature.

3 digital certificate[3]."

2.3.1.2 Portable Executable (PE) Section

Authenticode is used to sign Portable Executable (PE) files. When a PE file is signed, the Authenticode signature (*authentihash*) is embedded in the PE file: specifically, in *Attribute Certificate Table* specified in the *Certificate Table* entry in *Optional Header Data Directories*. An *authentihash* excludes certain PE sections in

[3] `https://submit.symantec.com/whitelist/`

the yellow background in Figure 2.2.

2.3.2 Code Signing Process

Similar to the Web PKI (e.g., TLS), a publisher (i.e., a software author) first applies for a code signing certificate from a CA with a pair of public and private keys, as illustrated in Figure 2.3. After verifying the publisher's identity, the CA issues a code signing certificate based on the X.509 v3 certificate standard [5]. Unlike TLS where a TLS certificate has a domain name as a common name, but a code signing certificate has a software publisher's name in a common name. In an X.509 v3 certificate, the applicant's (publisher's) information (such as locality address, company name) is bound to the publisher's public key. The publisher uses his/her private key associated with the certificate to sign his/her software. Specifically, in the signing process, the hash value of the software is first computed, and then, the hash value is digitally signed with their private key. In the end, the digital signature and the chain of the certificates are bundled with the original software. On the client-side, clients verify the signature with the public key embedded in the certificate when encountering the signed software. Clients are able to detect any modifications to the program.

2.3.3 Trusted Timestamping

One of the distinct differences from other PKIs (such as TLS) is *trusted timestamping* [37]. In the Web PKI, when a TLS certificate expires, the certificate is no

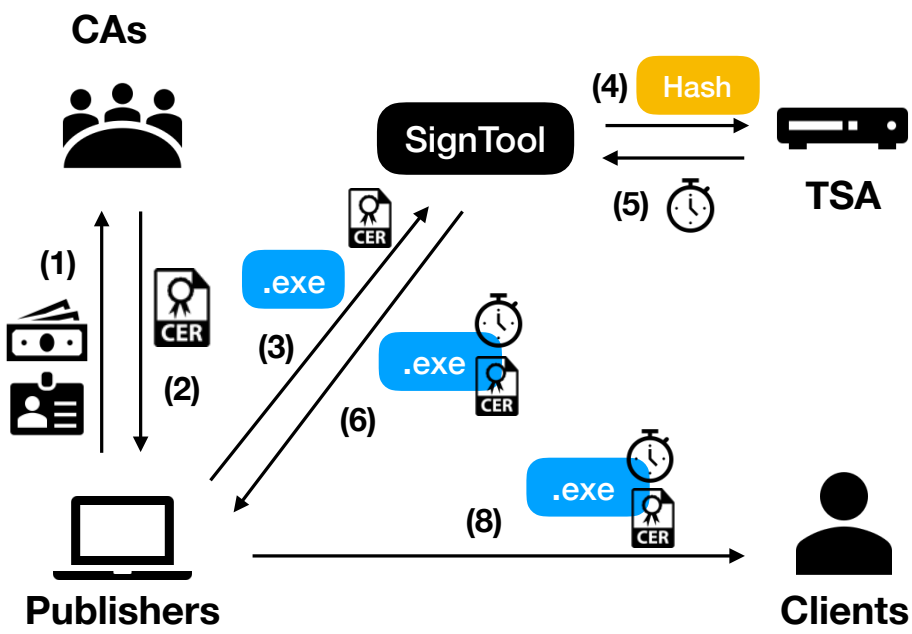

Figure 2.3: **Code signing process.** 1) A publisher apply for a code signing certificate from a CA with his/her identifications, 2) After vetting, the CA issues a code signing certificate to the publisher, 3) Using the *SignTool* (a signing tool provided by Microsoft), the publisher signs a binary sample with the certificate, 4) when a TSA is specified, the signing tool sends the hash value of the binary sample to the TSA server, 5) The TSA server issues the timestamp and signs the timestamp, and send them back to the signing tool, 6) The signing tool finally embeds the code signing and TSA certificate chain, digital signature, and timestamp into the binary sample, and 7) Finally, the publisher distributes the signed binary sample in the wild.

longer valid. Accordingly, the domain and web service bound to the certificate also become invalid. However, in the Code-Signing PKI, due to the trusted timestamping. a signed binary sample can be valid even after its code signing certificate expires if the binary sample file has been properly signed within the validity period of the code signing certificate. This is because other trusted parties (i.e., Time Stamping Authorities, TSAs) guarantee the authenticity and integrity of the timestamp by signing the timestamp with their TSA certificate when the binary sample file is signed. In turn, software clients are able to establish trust in the signed timestamp by the TSA using the same chain of trust similar to other PKIs.

2.3.3.1 Process

More specifically, a publisher is required to send a hash value of a binary sample file to a TSA; then the TSA signs the current timestamp and the hash value with TSA's certificate, and sends the current timestamp as well as the signature back to the publisher as shown in Figure 2.3. The publisher bundles the binary sample file with the timestamp signature and TSA's certificates. Clients verify the signature with the TSA's public key in the TSA's certificate.

2.3.4 Revocation in the Code-Signing PKI

Private keys can be compromised due to several reasons: for example, private keys are stolen. The primary defense against compromised certificates is *revocation*. Not only issuing code signing certificates, but also revoking compromised certifi-

cates are the CAs' main roles. Unlike the Web PKI where the compromised TLS certificates immediately become invalid when CAs revoke the compromised TLS certificates, in the Code-Signing PKI, CAs must investigate when code signing certificates become compromised, and set a revocation date, called *effective revocation date*.

An effective revocation date determines the validity of signed binary files. In other words, even though a code signing certificate is revoked, binary sample files signed with the code signing certificate can be valid depending on the effective revocation date set by an issuing CA. As shown in Figure 2.4, suppose that a code signing certificate (issue date: t_i and expiration date: t_e) is used to sign five binary sample files at $t_{b1..2}$ and $t_{m1...3}$ (two of them are benign binary samples and the others are malware). At a certain point, the certificate is found to be compromised. The issuing CA revokes the compromised certificate and set *effective revocation date* at t_r between t_{m1} and t_{m2}. The binary samples signed at t_{m2}, t_{m3}, and t_{b2} are no longer valid since the these signing dates are after the effective revocation date, t_r. However, the binary samples signed at t_{b1} and t_{m1} (including malware) can be still valid even though the certificate is already revoked since t_{b1} and t_{m1} are earlier than t_r.

There are two types of revocation dates: 1) *effective revocation date (t_r)* and 2) *revocation publication date (t_p)*. Effective revocation date determines the validity of signed program files. Revocation publication date is when CAs disseminate the revocation information clients (i.e., adding revoked serial numbers into CRLs).

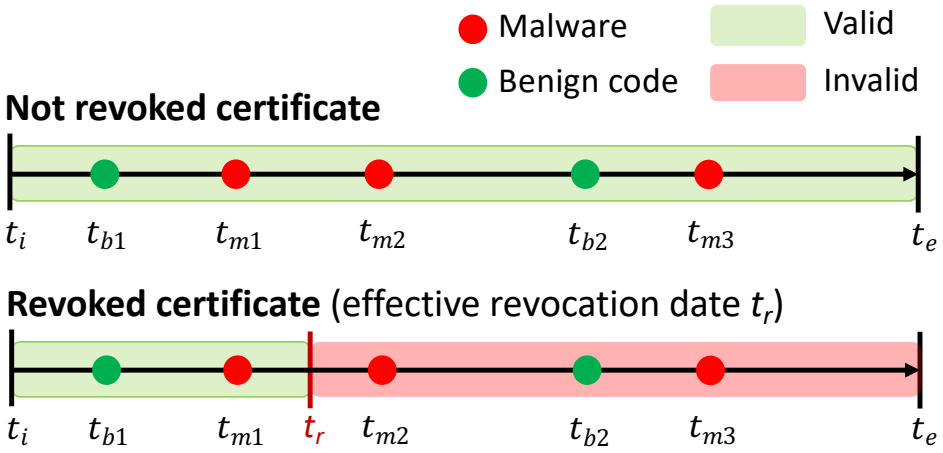

Figure 2.4: **1) Example of an effective revocation date (t_r).** It determines the validity of signed malware. 2) **Revocation delay (t_p - t_d).** (t_i: issue date, t_e: expiration date, t_r: effective revocation date, t_b: signing date of a benign program, t_m: signing date of malware, t_d: detection date, and t_p: revocation publication date). When an effective revocation date is set to t_r, the malware signed at t_{m1} can be valid even though its code signing certificate has been already revoked because it was signed before t_r. [4]

2.3.4.1 Two Policies of Setting Effective Revocation Dates

There are two types of the revocation policies in the Code-Signing PKI; 1) *hard revocation* and 2) *soft revocation*. When the effective revocation is set to the issue date of a certificate, it is called hard revocation, and the effective revocation date is set between t_i and t_e, which is soft revocation. In *hard revocation*, all signed binary samples (including malware and benign binary samples) become invalid so that there exist no longer security threats. However, the software publishers have to re-sign

their benign software and re-distribute the signed software. In *soft revocation*, if CAs know when certificates become compromised, they need to set effective revocation dates between between t_i and t_e. In this case, they do not need to re-sign their all software.

2.3.4.2 Dissemination of Revocation Information

After revoking compromised certificates, CAs must disseminate the revoked certificate information to clients. The two predominant ways to disseminate certificate revocation information are (1) Certificate Revocation List (CRL) [5] and (2) Online Certificate Status Protocol [28].

CRLs contain the revocation information (certificate serial numbers, (effective) revocation date, revocation reason) of certificates that have been revoked. Each CRL is updated based on their CA's issuance policy; for example, they can be issued when a newly revoked certificate is inserted, or a specific time of day or a day of a month. The location of the CRL for a specific certificate is contained in the signed binary at CRL Distribution Point (CDP) of the X.509 certificate. The location of the CRL is specified at CRL Distribution Point (CDP) of the X.509 certificate. Clients have to periodically download the entire CRL (not just recent changes) to check the latest revocations. HTTP is predominately used for downloading CRLs.

OCSP was introduced to resolve the network overhead problems of CRL. Clients can simply query an OCSP server for a certain certificate, which helps mitigate the network overhead at the server as well as clients. Authority Information

Access (AIA), an extension field in an X.509 certificate specifies OCSP point for each certificate.

The TLS CAs are typically not responsible for providing the revocation status of expired certificates. The code signing CAs, however, must maintain and provide the revocation information of all certificates that they have issued including expired certificates due to the trusted timestamp [38, 39]. Since the trusted timestamp extends the life of a signed binary, CAs must maintain the CRLs and OCSP in perpetuity to make revocation information always-available for clients.

Chapter 3: Related Work

In this chapter, I discuss related work in two key areas: measuring the ecosystem of the Web PKI (TLS) certificates; measuring Windows (*Authenticode*) code signing abuse.

3.1 Measurements of the TLS Certificate Ecosystem

3.1.1 Data Collection of TLS Certificates

One of the most representative examples in PKIs is the Web PKI (i.e., TLS). The ecosystem of the Web PKI has been well studied and the security threats inherent in the ecosystem have been identified, which has helped the improvements of the ecosystem. This is because many network scanners such as Rapid7 [40] and ZMap [41] had been introduced and used to collect TLS certificates. In particular, ZMap is able to scan the entire IPv4 address around less than one hour, which enables TLS certificate researchers to readily obtain a large number of TLS certificates. Moreover, as mentioned in Section 2.2.5, CT becomes a de-facto data source for TLS certificate research because most TLS certificates are logged after issued [31]. This is the main distinct difference from the Code-Signing PKI where collecting code

signing certificates in the wild is significantly challenging since signed executable binaries and its code signing certificates cannot be readily located.

3.1.2 Identifying Security Problems in the Web PKI

Using the TLS certificates collected by Rapid7 and ZMap, Durumeric et al. [42] have uncovered the bad practices on leaf TLS certificates and CA (intermediate) certificates that can impact on the security of the HTTPS ecosystem. For example, 50% of trusted leaf TLS certificates have an insecure 1024-bit RSA key. Also, CAs still sign their issued TLS certificate using MD even in April 2013.

There was a significant critical vulnerability in *OpenSSL* that can leak private keys of certificates, called *Heartbleed*. Two measurement studies [43, 44] regarding the impacts of the *OpenSSL* vulnerability on the HTTPS ecosystem. They found that 11% of HTTPS sites in the Alexa 1 Million websites still have the *HeartBleed* vulnerability, two days after disclosure. Moreover, 14% re-isssued TLS certificates still have the same the private keys that were likely leaked. Cangialosi et al. found that many websites share their private key with Content Delivery Networks (CDNs) or hosting providers, and they also manage their customers' private keys instead of the key owners [45]. Researchers have also focused on the revocation of TLS certificates. Liu et al. found that many revoked certificates were used, and web browsers failed to check the revocation of the certificates [46].

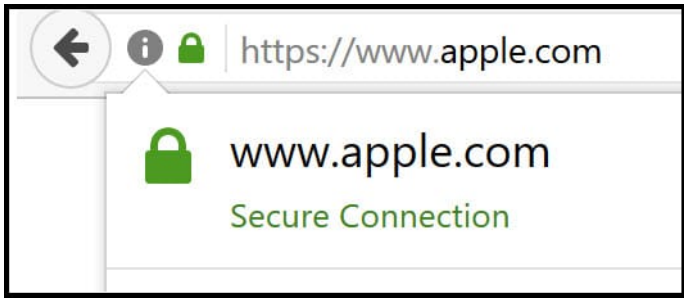

Figure 3.1: **Green padlock example.**

3.1.3 Phishing Attacks using TLS Certificates

3.1.3.1 Phishing Attacks

Phishing attacks are being widely used by adversaries to steal victims' credentials such as passwords and user IDs. Typically, adversaries mimic legitimate websites such as Paypal, Google, Facebook, etc. and lure a victim to input their credentials. Phishing attacks are often delivered to victims through phishing emails, social network message, and SMS [47–49].

Phishing attacks have been well studied: particularly, squatting phishing domains [50–53]. Specifically, in these phishing studies, they measured how many the squatting techniques have been exploited in the wild. Tian et al. [54] found that 657K domains (out of 224 million DNS records) that likely impersonated 702 brands using squatting phishing domain techniques.

3.1.3.2 Phishing Attacks with TLS Certificates

In most modern web browsers such as Chrome and Firefox, a padlock or green padlock icon is displayed for users to indicate the authenticity of a website that they access and their communicates are encrypted. However, users tend to misperceive the green padlock displayed in web browsers. They misunderstand that websites are *benign* and *legitimate* when a green padlock icon appears in web browsers.

Specifically, prior works [15–18] revealed that 1) even some users think HTTPS can protect against phishing attacks, 2) user believe that HTTPS means the trustworthiness of websites, and 3) also they perceive the green padlock as an indicator of trusted and safe websites.

These mis-perceptions and misunderstandings are able to help users to be tricked by phishing websites deployed with valid TLS certificates. Anti-Phishing Working Group (APGW) recently reported that approximately 60% of phishing websites are served with TLS certificates [19], which means that more phishing websites with TLS certificates appear in the wild than plain HTTP phishing websites.

However, HTTPS phishing attacks have been little studied. If we better understand the current landscape of HTTPS phishing attacks, we can mitigate the HTTPS phishing attacks as well as eventually improve the security of the Web PKI.

3.2 Measurements of the Code Signing PKI Abuse

Comparing to the Web's PKI, little is known about the code signing abuse. In the first attempt [10], *F-Secure* in 2010 introduced the ways to abuse Microsoft Authenticode. However, the work was not an officially peer-reviewed paper, rather presentation slides focusing on the introductions of new security threats in Authenticode.

Sophos also discussed the code signing abuse [11]. They measured digitally signed Windows PE malicious files collected from 2008 to April 2010. In this work, they found that the majority of signed malicious files were fake AV malware, followed by Spyware and Adware. The fake AV malware was signed with certificates legitimately issued from CAs.

Kotzias et al. [12] in 2015 examined 356,000 digitally signed executable files collected between 2006 and 2015. They observed that most of the signed malicious files are Potentially Unwanted Programs (PUPs), but signed malware is little found in their data sets.

Alrawi et al. [13] also conducted a measurement study of signed malicious files and malicious code signing certificates. In their work, they attempted to understand the abuse of code signing certificates misused to sign malware samples using 3 million malware samples. However, their analysis was based on all certificates even used not only for malware and for PUP or ADware.

Unlike the Authenticode code signing that obtains a certificate from a CA, Android applications are signed with self-signed certificates; thus, there is no effec-

tive revocation system. Many Android developers use the same key for their many applications, which can lead to unexpected security threats such as signature-based permissions [55].

Chapter 4: Understanding TLS (HTTPS) Phishing Attacks

In this chapter[1], we present how adversaries (i.e., phishing attacks) obtain TLS certificates from CAs for their phishing attacks and how the issuing CAs perform revocations against the compromised and/or misused TLS certificates.

4.1 Motivation

Transport Layer Security (TLS) [56, 57]—previously, Secure Sockets Layer (SSL)—was designed and deployed for secure communication in untrusted networks such as the Internet. As of April 2020, more than 90% Internet traffic is securely made over TLS [6], which means that most people use TLS for secure communication with other people or websites such as online banking services.

TLS relies on a Public Key Infrastructure (PKI) for the authenticity of entities whom others want to securely communicate, which called *the Web PKI*. In the Web PKI, besides clients and servers (i.e., service providers), there is also trusted third-parties, called Certificate Authorities (CAs) who issue TLS certificates after verifying applicants' identities or the ownership of domains. The TLS certificate includes an applicant's (i.e., server's) public key and binds the public key to the

[1]This paper is in submission [1].

server's domain. In turn, clients can establish trust in the issued TLS certificates by verifying certificates, and check if they access the domain specified in the verified TLS certificate.

In most modern web browsers such as Chrome and Firefox, a gray or green padlock icon is displayed for clients to indicate the authenticity of a website that they access and their communicates are encrypted. Technically, the green padlock icon means that the authenticity and integrity of a website that a user accesses are guaranteed. In other words, the icon is completely not related to the legitimacy or benignity of a website. However, many people tend to incorrectly believe that the meaning of a green padlock icon. They often misperceive that the websites are *benign* and *legitimate* if such a green padlock icon is displayed in the URL bars of their browsers. Prior works have shown that even worse, some users consider TLS (or HTTPS) as a defense against phishing attacks [15–18].

These misperceiving and misunderstanding are able to increase the chance for users to be tricked by phishing websites deployed with *valid* TLS certificates because the *valid* TLS certificate will be rendered as a green padlock icon in the URL bar. Particularly, Anti-Phishing Working Group (APWG) has reported that approximately more than 60% of phishing websites are launched with TLS certificates in the 1st quarter of 2019 [19]. Comparatively, in 2016, only less than 2% of phishing websites had TLS certificates. This indicates that over three years, the number of phishing websites with TLS certificates has significantly grown. Therefore, now more phishing websites with TLS certificates appear in the wild than the plain HTTP phishing websites. In this regard, we need to better understand the

current landscape of TLS phishing attacks, which leads to the enhancement of the security of the Web PKI.

In this chapter, we attempt to understand the current landscape of HTTPS (i.e., TLS) phishing threats. We first analyze 5.05M phishing URLs collected from eCrimeX, a blacklist of phishing URLs operated by Anti-Phishing Working Group (APWG). Next, we conduct investigations on the usage of TLS certificates in phishing websites and the ecosystem of the TLS CAs. The collected dataset provides two insights on TLS certificates used for phishing attacks in the wild: (1) CAs rarely revoke abusive certificates and (2) a CA's vetting process can fail. Two organization validation (OV) certificates are issued to adversaries and misused in phishing attacks.

4.2 Landscape of TLS (HTTPS) Phishing Websites

In this section, we study TLS-based phishing websites to better understand how adversaries abuse the Web PKI (TLS and HTTPS) by deploying their phishing websites with TLS certificates.

4.2.1 Dataset Collection

We collect 5.05 Million phishing URLs from a phishing website blacklist website, operated by Anti-Phishing Working Group (APWG) [58]. This observation started on March 12th, 2017 and ended on October 1st, 2019, totally 30 months.

URLs	FQDN			Certificates		
	Distinct	HTTPS	GSB	Total	Valid	Expired
5,053,705	675,387	301,555	67,830	216,587	90,935	125,652

*Expiration and revocation status were checked on Oct. 1st, 2019.

Table 4.1: **Summary of TLS Phishing website datasets.** We collect HTTPS phishing URLs and certificates from Mar. 12th, 2017 to Oct. 1st, 2019 (30 months). *HTTPS* denotes the total number of phishing FQDNs deployed with HTTPS. *GSB* denotes the number of distinct domains blacklisted in Google Safe Browsing. *Valid* denotes that certificates are valid in terms of validity period; thus, even revoked certificates also belong to this category [1].

4.2.1.1 Collections of Abusive TLS certificates

The simplest way to obtain TLS certificates is accessing websites and downloading the TLS certificates from the websites. However, because most phishing websites have ephemeral lifetime after blacklisted [59], the simplest way is unavailable at the time of accessing the blacklisted phishing websites and downloading their TLS certificates from the phishing websites. Therefore, Certificate Transparency (CT) is utilized. CT is designed and deployed for auditing purposes when TLS certificates are issued [60] so that CT logs TLS certificates immediately after they are issued. This means that all abusive TLS certificates misused for phishing

websites are able to be obtained even though these phishing websites are no longer alive.

We first extract distinct domain names from the 5.05M phishing URLS blacklisted in APWG. 675.4K distinct Fully-Qualified Domain Names (FQDNs) are found in the 5.05M URLs as described in Table 4.1. Of them, we observe that 301.6K (44.6%) FQDNs are deployed with the HTTPS protocol, which means they use TLS certificates. We then query CT with the distinct FQDNs and obtain 216.6K (71.8% out of 301.6K) X.509 TLS certificates from the CT logs.

The free web hosting services such as *000webhostapp.com* and the free online form builder service such as *typeform.com* are not observed in the CT logs. This is because these free online services utilize wildcard TLS certificates (e.g., **.000webhostapp.com*). Therefore, all sub-domains belonging to the free online service domains are not logged in the CT logs.

Only 42.0% TLS certificates are valid on October 1st, 2019. In other words, 58.0% TLS certificates have already expired and are no longer valid. This is mainly because automated CAs (ACME CAs such as Let's Encrypt and cPanel) issue TLS certificates whose lifetime is 90 days. Surprisingly, only 2.3% certificates have been revoked among these valid TLS certificates (more details are discussed in Subsection 4.2.5).

4.2.2 Google Safe Browsing

Google Safe Browsing (GSB) is the most prevalent anti-phishing technique, operated by Google. This technique is by default enabled in Google Chrome, Apple Safari, and Mozilla Firefox. When a user accesses a phishing website blacklisted by GSB, GSB blocks the access to the phishing website and displays a red warning webpage to the user.

We observe that only 17.1% phishing URLs (861,022 URLs out of 5.05M URLs) are blacklisted in GSB; the most phishing URLs are not blacklisted in GSB. In other words, users can still be exposed to the most phishing URLs (approximately 83% phishing URLs) although these phishing URLs have already been publicly available in the wild, which essentially echoes the observation in prior work [61]. In the study, they observed that the blacklist such as Google Safe Browsing mechanism is often ineffective.

Google Safe Browsing sometimes blacklists only URLs, not the FQDNs of the URLs. In the dataset, 124,260 distinct FQDNs are extracted from the blacklisted 861,022 phishing URLs. We again query Google Safe Browsing with the distinct 124,260 FQDNs, and find that only 113,344 of them are blacklisted. This means that 10,916 FQDNs (8.8% out of 124,260 FQDNs) are not blacklisted while its URLs are blacklisted.

4.2.2.1 HTTPS vs. HTTP

We compare the HTTPS phishing FQDNs with the plain HTTP phishing FQDNs. We find that HTTPS phishing FQDNs are more likely blacklisted in Google Safe Browsing then the plain HTTP phishing FQDNs. Specifically, only 22.5% (67,830) FQDNs of the distinct 301K HTTPS FQDNs are blacklisted in Google Safe Browsing while only 12.3% (45,814 out of 374K FQDNs) plain HTTP FQDNs are blacklisted in GSB.

4.2.2.2 Revocation

We query CT with the 67,830 FQDNs and find that 36,623 (54%) TLS certificates in the CT logs. We then check the revocation statuses of the 36,623 TLS certificates; 46.9% (17,194) TLS certificates already expired. Only 2.7% (519 out of 19,429) TLS certificates are explicitly revoked. In short, the domains blacklisted in GSB are hardly revoked and even though this blacklist information is publicly available, CAs may not utilize Google Safe Browsing to monitor their issued TLS certificates and check whether their TLS certificates are misused. More details are discussed in Subsection 4.2.5.

4.2.3 Squatting Domains

The primary goal of phishing attacks is to deceive users by impersonating the benign, legitimate webpages of online banking services, government departments, etc. Domain name squatting is one of the common impersonation techniques. Specif-

ically, adversaries leverage this technique to make victims confused with currently popular brands [52, 53]. For example, *paypall.com* is a squatting domain for the legitimate *paypal.com*—observe the additional 'l' in the squatting domain.

We conduct a comparison of HTTP squatting domains and HTTPS squatting domains. To identify squatting domains, We utilize the techniques from a prior work [50]. Then, we further measure how many squatting domain names exist in the domain names that use TLS certificates for HTTPS by comparing to domain names with HTTP. Particularly, squatting domain names typically share many overlapping letters with targeted legitimate domain names. We observe that only 3.81% of the domains with HTTPS (out of 378,747) are the squatting domains.

4.2.4 OV & EV Certificates

Recall that there are typically three types of TLS certificates: Domain Validation (DV) certificates, Organization Validation (OV) certificates, and Extended Validation (EV) certificates. DV TLS certificates are issued only after CAs check the ownership of domains that will be included in certificates. However, OV and EV TLS certificates need more strict applicants' vetting process because their organization information such as company name is specified in OV or EV TLS certificates. Therefore, CAs should verify domain ownership as well as business registration using certified documents such as government-issued business registration documents. This means that due to the strictness, adversaries (e.g., phishing attackers) have more challenges to obtain OV or EV TLS certificates than DV certificates. How-

ever, if they obtain OV or EV certificates, the phishing websites with the issued OV or EV certificate can increase trust for victims (clients) by providing the victims with the legal information of organizations.

4.2.4.1 OV & EV Certificate in Phishing Websites

Adversaries may want to launch phishing websites with OV or EV TLS certificates because these websites with more trustful certificates can present a higher level of trust to phishing victims. In turn, victims can readily establish trust in the phishing websites with OV or EV TLS certificates since these websites display more trustful icon and organizations' information (such as company name) to the victims.

However, obtaining OV or EV TLS certificates from CAs is significantly challenging for adversaries (i.e., phishing attackers). This is because the issuance of OV or EV TLS certificates requires more strict vetting process. In other words, adversaries need to submit their government-issued documents regarding company or organization information. For example, they need to incorporate shell companies to pass the strict vetting process, which requires more time and effort such as finances.

We raise a research question that how many OV and EV TLS certificates are misused for phishing attacks. We observe that totally 598 TLS certificates are misused for phishing websites in the dataset (Table 4.1); specifically, 294 OV and 304 EV TLS certificates. Most TLS certificates are blacklisted because benign web servers had become compromised and phishing websites are co-hosted in the compromised web servers after being exploited vulnerabilities. In other words, the

OV or EV certificates are not intentionally issued for phishing attacks.

Besides the compromised web server attacks, we find that two OV certificates in the dataset may be issued directly to adversaries. The two OV TLS certificates are misused in *Paypal* phishing attacks, which was confirmed by *Paypal*. One of the two OV TLS certificates contains a *Paypal* squatting domain in the Subject Alternative Names (SAN) filed and another OV certificate has a *Paypal* squatting domain as a common name.

Other squatting domains found in the SAN field from the OV TLS certificates are Google and Facebook as seen in Table 4.2. The parent domain was registered in January 2015. We find that 119 phishing URLs that contain the domain in the dataset. The parent domain contains multiple sub-domains such as "*hrmy.mtbank*", "*saayantan*", "*aruntest*," which are followed by "*paypal.com*", "*google.com*", "*facebook.com*"—e.g., *paypal.com.hrmy.mtbank.###.net* (the parent domain is masked with ###).

4.2.4.2 Threat Model for Abusive OV TLS Certificate

To better understand the adversaries behaviors—for example, how they leverage OV TLS certificates—we query CT again for the parent domain (*###.net*). Additional 4,843 OV certificates under the same the parent domain have been issued with numerous sub-domains. The first issuance of the OV TLS certificates happened in January 2015 and the latest issuance was in October 2019. This indicates that the adversaries have kept exploiting the CA after the parent domain was registered

Certificate #1	Certificate #2
paypal.com.saayantan.####.net	hrmy.mtbank.####.net
paypal.com.paypal.com.saayantan.####.net	www.facebook.com.hrmy.mtbank.####.net
*.www.google.com.paypal.com.saayantan.####.net	*.googleads.g.doubleclick.net.hrmy.mtbank.####.net
*.nexus.ensighten.com.paypal.com.saayantan.####.net	*.s0.2mdn.net.hrmy.mtbank.####.net
*.www.paypalobjects.com.paypal.com.saayantan.####.net	*.staticxx.facebook.com.hrmy.mtbank.####.net
*.paypal.com.paypal.com.saayantan.####.net	www.paypal.com.hrmy.mtbank.####.net
www.google.com.paypal.com.saayantan.####.net	*.www.paypal.com.hrmy.mtbank.####.net
www.paypalobjects.com.paypal.com.saayantan.####.net	*.www.facebook.com.hrmy.mtbank.####.net
nexus.ensighten.com.paypal.com.saayantan.####.net	*.tpc.googlesyndication.com.hrmy.mtbank.####.net
www.paypal.com.paypal.com.saayantan.####.net	googleads.g.doubleclick.net.hrmy.mtbank.####.net
*.www.paypal.com.paypal.com.saayantan.####.net	tpc.googlesyndication.com.hrmy.mtbank.####.net
*.paypal.com.saayantan.####.net	s0.2mdn.net.hrmy.mtbank.####.net
	hr.my.hrmy.mtbank.####.net
	*.hr.my.hrmy.mtbank.####.net
	staticxx.facebook.com.hrmy.mtbank.####.net
	*.hrmy.mtbank.####.net

Table 4.2: **X.509 v3 Subject Alternative Names (SAN) in the two OV TLS certificates.** The parent domain name is masked as #### [1].

in January 2015. Every OV TLS certificate has a number of Subject Alternative Names (SAN) for their phishing attack targets: for example, *outlook.com*, *box.com*, *google.com*, *facebook.com*, *slack.com*, etc.

Moreover, we investigate how the adversaries can obtain these numerous abusive OV TLS certificates. The easiest way for them is to pay for the all OV TLS certificates, but it requires quite a bit money. For example, the OV certificates have

been issued from a certain CA, and the CA charges $349 and an additional $99 for each sub-domain in the SAN field. As described in Table 4.2, the abusive OV TLS certificate #1 has eleven SANs, and in total, the single OV certificate costs $1,438. Therefore, they are unlikely motivated to purchase the expensive OV TLS certificates using their own money. Therefore, we suspect that they may use their own stable supply chain or exploit some vulnerabilities of the CA vetting process. This is because if they purchased the all OV certificates out of their own pocket, they need to totally pay approximately $4.8M (4,843 abusive OV TLS certificates $*$ $1,000) when an OV certificate with SANs is purchased at $1,000[2].

We present a few feasible scenarios regarding how the adversaries obtain the abusive OV TLS certificates. First, the phishing attackers may utilize stolen financial information such as stolen credit cards. However, as mentioned before, the total cost of the all abusive OV certificates is approximately $4.8M. It is very difficult that such financial transactions can be successfully made without being detected by banks since the total cost is tremendous. Furthermore, it is almost impossible to pass every vetting process where humans are involved to check applicants' identities; totally, the adversaries need to pass the 4,843 strict vetting process since each OV TLS certificate requires a human-involved vetting process. Second, another feasible scenario is that the adversaries compromise the CA's issuance process including the vetting process and/or exploit some vulnerabilities in the issuance process.

While being unclear how the adversaries obtain the 4,843 OV certificates for their phishing attacks, it is the fact that the CA obviously failed to check high-risk

[2]Note that that is an underestimation

certificate requests that include the domain names of target attacks such as *paypal.com*. Moreover, these OV TLS certificates have been not revoked yet, which indicates that the CA also failed to proactively discover their issued TLS certificates that are misused for phishing attacks. Also, users still remain exposed to the phishing attacks with the OV TLS certificates that increase the trust of phishing websites.

4.2.5 Revocation of TLS Certificates for Phishing Websites

Revocation is the final defense against the abusive TLS certificates; the issuing CAs make the compromised or abusive TLS certificates no longer valid. We check the revocation statuses of the all certificates in the datasets that are misused for phishing attacks.

There are two predominant methods to disseminate revocation information to clients as discussed in Subsection 4.2.5: OCSP and CRLs. OCSP is always preferred over CRLs in terms of checking revocation status so that we first check the revocation statuses of the abusive TLS certificates using OCSP after we filter out expired TLS certificates. However, we occasionally experience that the OCSP responders were unavailable or provided OCSP responses as prior work [62, 63] has revealed that OCSP responders are improperly managed and operated. In this case, CRLs are used as a complement way to retrieve revocation information. If both are unavailable, we consider them as *error*.

4.2.5.1 Revocation Rate

Averagely, only 2.3% of TLS certificates in the dataset misused for phishing attacks have been revoked as described in Table 4.3. This means that CAs hardly perform revocations for the abusive TLS certificates.

Interestingly, there is a distinct pattern between automated (ACME) CAs (e.g., Let's Encrypt and cPanel) and commercial CAs (e.g., DigiCert, Comodo, GoDaddy, etc.). Specifically, the automated CAs have lower revocation rates comparing to commercial CAs; GoDaddy has the highest revocation rate (31.4%).

The extremely lower revocation rate indicates that CAs rarely investigate their issued TLS certificates that are being misused for phishing attacks. As a result, users still remain exposed to phishing websites with TLS certificates because they are not revoked yet.

CA	Certificate	Status			Revc. rate
		Good	Revoked	Expired	
Let's Encrypt	110,670	40,931	64	69,632	0.16%
cPanel	71,890	22,893	111	48,866	0.48%
Comodo	11,136	6,114	328	4,694	5.09%
CloudFlare	10,509	10,092	0	416	0.00%
Go Daddy	5,286	3,122	1,430	734	31.41%
DigiCert	2,251	1,911	16	324	0.83%
RapidSSL	622	497	5	120	1.00%
TrustAsia	669	603	1	65	0.17%
GlobalSign	617	502	11	104	2.14%
GeoTrust	571	458	8	105	1.72%
Etc.	2,366	1,630	144	592	8.12%
Total	216,587	88,817	2,118	125,652	2.34%

*Expiration and revocation status were checked on Oct. 1st, 2019.

*64 certificates are excluded due to OCSP response errors.

Table 4.3: **Revocation statuses of top-ten CAs' certificates.** 59.6% certificates already expired as of Oct. 1st, 2019. Of the valid 40.4% certificates, on average, only about 2.3% certificates have been revoked. Interestingly, only Go Daddy revoked about 31% of certificates used for phishing websites. Other CAs barely revoked certificates [1].

Chapter 5: Understanding Adversary Behavior and Security Threats in the Code-Signing PKI

In this Chapter[1], we attempt to study the security threats in the Code-Signing Public Key Infrastructure (PKI). We begin with the motivation about why we need to study the Code-Signing PKI (Section 5.1). The Code-Signing PKI has an inherent challenge of collecting code signing certification. To overcome the challenge, we utilize multiple datasets that can be representative such as WINE (Section 5.2). First, we fully characterize the weaknesses of the Code-Signing PKI that adversaries potentially are able to exploit; specifically, we mainly focus the three actors—certificate authorities, software publishers, and end-users (Section 5.3). Them we try to understand the underground economy of code signing certificates (Section 5.4) Finally, we measure the effectiveness of the primary defense, revocation against the Code-Signing PKI abuses (Section 5.5). We made the results and datasets publicly available at *signedmalware.org*.

[1]Published as [2–4]

5.1 Motivation

The establishment of a trust in software distributed over the Internet is challenging due to the nature of software distribution; 1) we are unable to identify publishers and 2) there is a chance for the software to be tampered during distribution. In other words, the software can be readily altered for malicious purposes by adversaries such as inserting a malicious payload into the legitimate software.

To guarantee the authenticity and integrity of software, the digital signature mechanism is utilized, called *code signing*. A software publisher signs her/his software to distribute with her/his own private key never shared with others. In turn, a client verifies the digital signature with the public keys associated with the private key.

Another problem inherent in *code signing* is how we can attest that the public key associated with the private key legitimately belongs to the software publisher. To resolve this issue, code signing relies on Public Key Infrastructure (PKI), called the *Code-Signing PKI*. Similar to other PKIs such as the Web's PKI, the Code-Signing PKI requires Certificate Authorities (CAs) to certify a code signing certificate belongs to a legitimate publisher.

The CAs issue code signing certificates after verifying the publishers. The publishers use the issued code signing certificates to sign their software to provide the integrity and authenticity of their software. Clients are able to establish trust in the signed software by verifying the signed software with the public key embedded in the code signing certificates. Then, they can know who publishes the software

and the software is never altered after signed.

Unfortunately, anecdotal evidence has proved that the software properly signed by legitimate publishers can be critical malware. For example, *Stuxnet* [14, 64] is malware properly signed with two private keys of the two reputable Taiwanese semiconductor companies. The private keys were likely stolen to sign the malware since the two companies are very reputable and never believed to be involved in the malicious activity. The code signing certificates of the two reputable companies helped remain undetected for a longer period than other malware [14]. Another example is the *Flame* malware properly signed with a Microsoft code signing certificate. However, the certificate was not legitimately issued from a CA; rather it was a counterfeit code signing certificate generated by exploiting the MD5 chosen-prefix collision attack [65]. Moreover, a fraudulent certificate was issued by a CA due to CAs' verification failures. In 2001, *VeriSign* erroneously issued two code signing certificates under the name of Microsoft Corporation to an adversary who claimed to be an employee of Microsoft [66].

However, little research on the code signing abuse has been conducted; therefore, the security threats and challenges inherent in the code signing PKI are not understood well. Rather, prior work focuses on Potentially Unwanted Programs (PUPs) signed with code signing certificates issued legitimately from CAs [14,64,65]. In particular, 1) weaknesses in the code signing PKI are not systemically examined and how these weaknesses can lead to the breaches of the trust in the code signing PKI. Second, 2) there is no understanding of the adversaries' behaviors; specifically, how adversaries obtain code signing certificates misused for malware (Section 5.4).

Lastly, 3) the primary defense against these abuses, revocation is not fully understood.

In this chapter, we conduct a measurement study of these challenges and security threats inherent in the Code-Signing PKI. Particularly, to answer the first challenge, we devise a system and an algorithm to identify the weakness in the Code-Signing PKI, and how this weakness can lead to the breaches of the trust in the PKI (see Section 5.3). To better understand the malware authors and black market vendors, we measure the black market for code signing certificate (see Section 5.4). Lastly, the primary defense against the code signing abuse is revocation. To see if the revocation is effective, we examine the current revocation process (see Section 5.5).

5.2 Overview of the Data Sources

In this section, we provide an overview of the data collection for measurement studies of the code signing PKI abuses: such as how we collected the dataset and what the challenges are inherent in for the code signing certificate collection.

5.2.1 Worldwide Intelligence Network Environment (WINE)

WINE [67] is a platform that collects data from about 10.9 million real end-hosts around the world who have installed Symantec products and opted in the agreement for sharing their data with Symantec; but, identifiable clients information has been not collected. Particularly, the collected datasets are clients' download and installation activities.

From the WINE dataset, we extract 1) the hash values (SHA256), 2) the server-side event timestamps, 3) the publisher names of the downloaded/installed executables on the end-hosts. However, the WINE dataset does not include more information than the extracted ones such as code signing certificates information. Therefore, we are unable to distinguish between executable files with the same publisher names; particularly, when executable files are signed with different code signing certificates that belong to the same publishers. To overcome this challenge, we utilize another service that provides more detailed information about executable files, called VirusTotal (Section 5.2.2.1). The WINE dataset is used for Section 5.3 and Section 5.5.

5.2.2 Ground Truth Data

5.2.2.1 VirusTotal

VirusTotal[2] is a web service that allows us to scan executable files with up to 70 anti-virus (AV) engines, and also provides us with more detailed information of scanned executable files such as code signing certificate information.

As mentioned in Section 5.2.1, the WINE dataset does not provide the code signing certificate information such as serial numbers, issue dates, expiration dates, etc. To collect this information about code signing certificates used to signed executable files in the WINE dataset, we query VirusTotal [68]. Additionally, we also query VirusTotal for the first-submission timestamps to VirusTotal, the number of

[2] https://www.virustotal.com/

AV engines that classified the executable files as malware, the labels of malware.

5.2.2.2 OpenCorporates

OpenCorporates[3] is a website that provides the largest open database of businesses in the world; approximately 100 million companies. We use this open database to check if the publishers found in malicious code signing certificates are legitimate (Section 5.3).

5.2.2.3 HerdProtect

To better understand the reputation of publishers (software companies), we utilize *HerdProtect*[4]; particularly, the publisher information (location, business type, etc.) and whether the publisher has been released Potentially Unwanted Programs (PUPs) are queried.

5.2.3 Code Signing Certificates

The code signing certificate information provided from VirusTotal (Section 5.2.2.1) includes only issue dates, expiration date, serial numbers, and issuers (CAs). This information, however, is not enough for the measurement study of revocation (Section 5.5) since the measurements in this work require the information of the dissemination channels for revocation status. Therefore, for more information from code signing certificates, we collect code signing certificates used in the wild.

[3] `https://opencorporates.com`

[4] `http://www.herdprotect.com/`

One of the main challenges in collecting code signing certificates is that there are no publicly available repositories or datasets; while in TLS, *censys.io* provides the TLS certificates or TLS certificates can be readily collected through network scanners such as ZMap [41]. To overcome this challenge, we utilize multiple data sets publicly released from prior research, *Malcert* [13], and *Malsign* [12], and Symantec's proprietary repository of binary samples.

5.2.3.1 Revocation Information

As discussed in Section 2.3.4, the revocation status information is mainly disseminated over two mechanisms: 1) CRLs and 2) OCSP. Clients have to access CAs' dissemination servers through the URLs of the two mechanisms specified at *CRLDistributionPoints* and *AuthorityInfoAccess* extensions respectively in x509 v3 code signing certificates [23]. We extract CRLs and OCSP URLs from the 145,582 leaf code signing certificates. Most certificates (137,027, 94.1%) have both CRLs and OCSP URLs; only CRLs URLs are specified in 7,794 (5.3%) code signing certificates and only OCSP points are embedded in 98 (0.06%) certificates. However, the CRLs can be used for other purposes such as TLS; thus we manually remove them, which remains only CRLs used for code singing PKI.

5.2.3.2 Revocation Publication Date List

A CRL file includes the serial numbers of revoked code signing certificates, revocation date, and revocation reason. The revocation date is rather an effective

revocation date (t_r), not revocation publication date. The effective revocation date determines the validity of signed executable files.

To collect the revocation publication date(t_p), we devise a system, called revocation publication date collection system. The system collects revoked serial numbers once a day from the CRLs data set. This information can be utilized to measure the revocation delay, how long CAs take to revoke compromised certificates.

5.3 Understanding of Code Signing Abuse

Compromised code signing certificates have used for advanced threats; for example, *Stuxnet* that included device drivers signed with likely stolen private keys stolen from two Taiwanese semiconductor companies [14]. *Flame* exploited a chosen-prefix collision attack against the MD5 hash [65]. Both examples indicate that the valid digital signature can help the malware evade detection and bypass AV engines.

Prior anecdotal information tells that many malware may carry valid digital signatures from compromised certificates [14, 64, 65]. However, these security threats have been little explored systematically. The previous research focuses on signed Potentially Unwanted Programs (PUPs), such as adware. The PUPs are typically signed with certificates legitimately issued from CAs. Therefore, the prior research does not distinguish between certificates legitimately issued to publishers and compromised certificates such as stolen certificates.

In this work[5], we conduct a systematic measurement study of threats that can breach the trust inherent in the Windows Code-Signing PKI. we focus on only signed malware, not PUPs.

In short, the contributions are the followings.

1. We proposed a threat model with three classes of weaknesses that an adversary can exploit (*1) inadequate client-side protections*, *2) publisher-side key mismanagement*, and *3) CA-side verification failures*).

2. We analyzed how long users are exposed to these security threats.

5.3.1 Measurement Methods

In this section, to better understand the code signing abuse, we describe 1) what data sources are used, 2) what the pipeline for this work is, 3) how to label signed malware, and 4) a new abuse detection algorithm.

5.3.1.1 Binary Labeling

It is important to classify and label binary samples and to distinguish signed malware from signed benign and signed Potentially Unwanted Programs (PUPs) since, in this work, we conservatively focus on only signed malware that exploits the code signing abuse.

For malware, we utilized the previous approach proposed in Kown's work [69]. In particular, for each binary sample, we define c_{mal} as the number of Anti-Virus

[5]Published as [2]

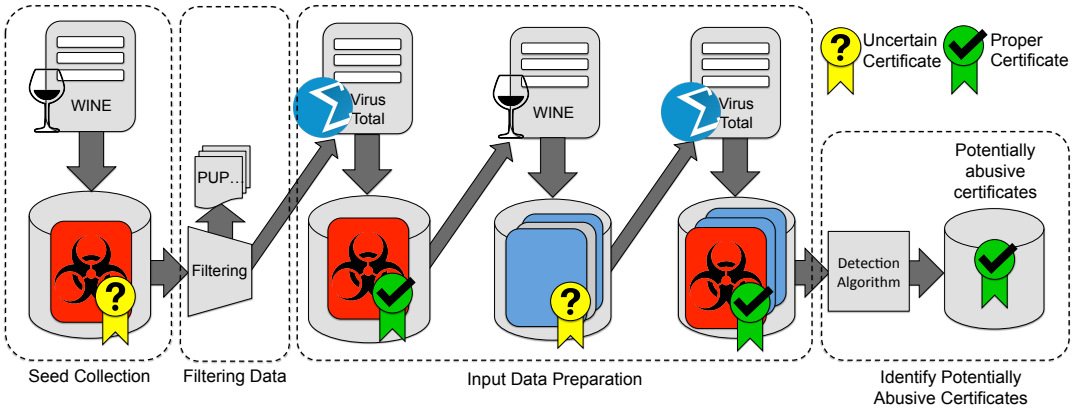

Figure 5.1: **Data analysis pipeline** [2]

(AV) engines that flag the samples as malware. we set $c_{mal} \geq 20$ as the threshold for malware. To be more conservative, we add another factor, r_{pup} that is the number of AV engines that detect as PUPs. In short, we consider a binary sample is malware if the sample has $c_{mal} \geq 20$ and $r_{pup} \leq 10\%$. For example, if a binary sample is detected as malware by more than 20 AV engines and another less than 10% of AVs detect it as PUPs, the binary is considered as malware.

Classifying benign binary samples is also required for the new abuse detection algorithm (Section 5.3.1.3). Similar to the approach for detecting malware, if a binary sample is $c_{mal} = 0$, we consider it a benign sample.

5.3.1.2 System Overview

As shown in Figure 5.1, the data collection and analysis requires four steps: 1) seed data collection, 2) filtering data, 3) input data preparation, and 4) identifying

potentially abusive code signing certificates.

Seed data collection. The data collection for this work starts with the unique SHA256 hashes of binary files from the AV telemetry data sets in WINE. We first exclude the hashes of binary samples that are not related to any malicious activities such as hacking tools or adware. This is because we have limited access to query VirusTotal with the hashes for the reports. We also add a list of hashes of known malicious binaries separately provided by Symantec. Then, we join this list with the binary reputation scheme in Symantec to figure out which binary files are digitally signed. This remains a list of potentially signed and malicious binary files.

Filtering data. The list of potentially signed and malicious binary files generated from the previous step may contain benign files or PUPs. Therefore, we first filter out PUPs using three ways, 1) we remove from the list the PUP publishers identified in prior work [12, 69–71]. 2) Then, we query *HerdProtect* (see Section 5.2.2.3) with all publisher names (i.e., common names in code signing certificates), and if the publisher names are found and classified as PUP publishers in the web service, we also exclude them. 3) Finally, we pick ten binary samples for each publisher name and filter out them if at least one of the binary samples is classified as PUPs as discussed in Section 5.3.1.1.

Input data preparation. We query VirusTotal with the filtered hashes, which returns VirusTotal reports that contain detailed information such as code signing certificates (serial numbers, issue date, and expiration date) and AV detection and labels for each binary samples. Binary samples are considered as signed malware

if 1) properly signed, and 2) the labels for the binary samples are malware as presented in Section 5.3.1.1. Note that in this step, there may be malware samples improperly signed with malformed signatures. After this step, we join the SHA-256 hashes with binary reputation scheme in WINE, which results in a list of potentially benign binary samples and malware samples that signed with the same code signing certificates. Then, we finally query VirusTotal again with this list, and we utilize the VirusTotal reports to identify benign and malicious samples.

Identifying potentially abusive certificates. In the last step, for each code signing certificates misused to sign malware, we use the abuse detection algorithm (described in the below Section) to identify how they are abused.

5.3.1.3 Abuse Detection Algorithm

Recall that in the third step (Section 5.3.1.2) of the pipeline where binary samples can carry malformed digital signatures. The malformed digital signatures can be obviously verified since the signatures do not match the hash values of the binary samples. Rather, the malformed signatures were likely copied from other signed binary samples. In this case, the adversaries do not have code signing certificates to sign their malware, but they attempt to make their malware samples look like being signed.

Besides the binary samples with malformed signatures, other samples identified by the pipeline may be properly signed with code signing certificates that can be still valid, expired, or revoked. These samples are malware and benign, but not PUPs.

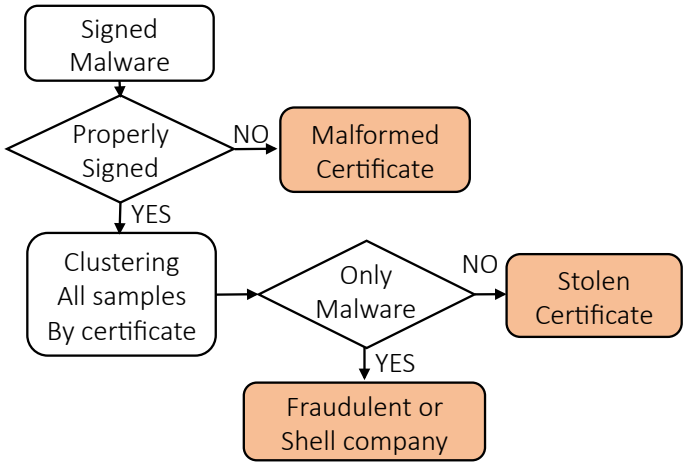

Figure 5.2: **Flowchart of the abuse detection algorithm** [2]. We consider code signing certificate are stolen when only signed malware is found in each cluster. Otherwise, when signed malware and benign executable binaries are found in each cluster, we consider them as the fraudulent or shell company case.

We first group the binary samples by each code signing certificates. A publisher can have multiple signed binary samples and the binary samples can be grouped into multiple pools. Each group may consist of 1) only signed benign samples, 2) only signed malware, or 3) both signed malware and benign samples. To expand the data set, we utilize *HerdProtect* and query the web service with each code signing certificate's serial number.

For each certificate misused for malware, we then infer the corresponding abuse types using the algorithm illustrated in Figure 5.2. There are three types of code signing abuses, and they are explained in detail.

Compromised certificates. A compromised code signing certificate is initially

issued to a legitimate, benign software publisher, and then the code signing certificate is misused to sign malware after the private key associated with the certificate is compromised such as stolen. Consequently, we expect the two types of binary samples are found in a certain certificate group; the two types are 1) benign binary samples signed and released by the legitimate publisher and 2) malicious samples signed and released by the malware authors who steal the private key. In each group, we further investigate the trusted timestamps to know the timeline of the abuse.

Identity theft & shell companies. In this case, malware authors obtain code signing certificates to sign their malware by convincing CAs. We believe that there is no motivation for them to sign benign samples and release signed ones. Therefore, we expect to see only signed malware found in a certain certificate's group. To distinguish between the two cases of identity theft and a shell company, we utilize *OpenCorporates* and *HerdProtect*. In particular, if a publisher is found in the two web services and its locality address in its code signing certificate does match the information in the web services, we suspect the publisher is a victim of identity theft.

Verification. We are able to reliably identify all signed binary samples with a malformed digital signature using the VirusTotal reports. we run an experiment to determine whether these signatures help bypass the protections for clients such as AV engines. Of all types of code signing abuses, we have a high degree of confidence in the determination that a code signing certificate is compromised since we are able to observe that the compromised certificate is likely utilized by multiple

Cert.	Desc. (Error code)	Total	Malware
Properly	Valid	130,053	109
	Revoked (0x800b010c)	4,276	43
	Expired (0x800b0101)	17,330	37
	Total	151,659	189
Malformed	Bad Digest (0x80096010)	1,880	101
	Others	81	0
	Parsing Error	233	35
	Total	2,194	136
Total		153,853	325

Table 5.1: **Property of the certificates** [2].

actors. However, for some compromised code signing certificates, we are not able to determine it because we cannot collect all benign binary samples in the wild. Similarly, we also have a higher degree of confidence in the identity theft than in shell companies since some publishers' information can be not found in the two web services (*OpenCorporates* and *HerdProtect*) we utilize. To be more conservative in verifying the results, we manually analyze their timeline and contact the publishers and CAs to confirm the findings.

5.3.2 Measurement Results

5.3.2.1 Summary of the Input Data

In this section, we summarize the input data in each step described in Section 5.3.1.2. Of 70,293,533 unique binary samples from the AV telemetry reports,

only 1,053,114 binary samples have the binary signer information in the binary reputation data set, which means that 1 out of 67 binaries are considered as signed. Note that this is the number of randomly sampled files since we have no visibility into all signed malware targeted on end-hosts in the world, and other AV engines may differently handle the digital signatures. However, WINE is the most representative data set collected by Symantec [72], one of the largest security vendors.

We first filter out potentially unwanted programs (PUPs) as described in Section 5.3.1.2, which remains 526,487 unique signed binaries (hashes). To be more conservatively, we further reduce the number of the signed binaries by filtering out 268,404 executables signed with the 2,648 certificates used for PUPs. After this step, this yields 258,083 signed binaries. we query VirusTotal with these binary's hashes; of these hashes, 88,154 hashes are not found in VirusTotal. we then remove 104,230 samples with a broken chain of code signing certificates, which remains 153,853 signed binary samples in the seed set. To use the abuse detection algorithm as described in Section 5.3.1.3, we also search for other PUPs in the data set, and we in total identify 415,377 signed binary samples found in VirusTotal.

As mentioned in Section 5.3.1.1, we set a very conservative number $c_{mal} \geq 20$ for the malware detection threshold, which helps identify only obvious malware samples. Moreover, this small number of signed malware can allow us to examine every single malware manually. (587 and 1136 binaries are classified as malware when the threshold is 10 and 5 respectively.) The number of properly signed malware per year for $c_{mal} \geq 5$ is presented in Figure 5.3.

We further validate the digital signatures of these binaries using the verified

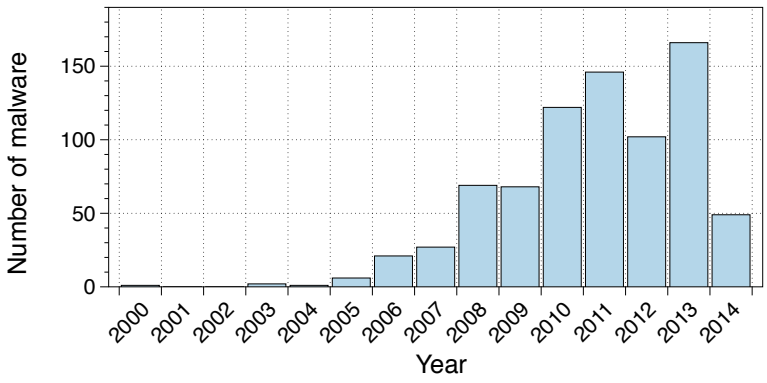

Figure 5.3: **Number of properly signed malware per year** ($c_{mal} \geq 5$) [2]

messages in the VirusTotal reports. VirusTotal checks the digital signatures using the *sigcheck* tool. For example, the sigcheck message, "a certificate was explicitly revoked by its issuer" indicates that the certificate is no longer valid since it is already revoked. This error corresponds to the error code "0x800B010C" in Microsoft Authenticode. Table 5.1 breaks down the validity status. Of 153,853 signed samples, the 325 samples are classified as signed malware. Of this 325 signed malware, 41.8% are improperly signed with malformed certificates while 58.2% binaries are properly signed. Most (74.3%) of improperly signed malware result from bad digests. Of 189 properly signed malware, 22.8% binaries are already revoked, and 19.6% have no valid trusted timestamps and expired certificates. Almost more than half of them (57.7%) are still valid.

CA	Count (%)
Symantec/Verisign	23,325,279 (60.47%)
Symantec/Thawte	7,054,263 (18.29%)
Comodo	2,059,601 (5.34%)
GlobalSign	708,618 (1.84%)
Go Daddy	704,036 (1.83%)
DigiCert	429,159 (1.11%)
Certum	48,677 (0.13%)
WoSign/StartCom	43,578 (0.11%)
WoSign	38,758 (0.10%)
Go Daddy/StarField	21,410 (0.06%)
Total	38,572,995 (100%)

Table 5.2: **The number of binaries signed with code signing certificates issued by each CA in WINE** [2].

5.3.2.2 Code Signing Ecosystem

In this section, we understand the code signing ecosystem using all signed binaries (both benign and malicious samples) the WINE data set. The numbers in WINE may be likely biased since all end-hosts run a Symantec product a least. Therefore, we remove the binaries signed with Symantec and then extract the issuer name of the digital code signing certificates from 38.6 million binaries. This allows me to have 210 unique CAs.

CA market share. We utilize the WINE data to better know the market share. Since we do not have the exact certificate of the signed binaries, we take an indirect

method to estimate the market share of the CAs: prevalence of the binaries signed by each CAs. We investigate the file signer information in the binary reputation dataset in WINE. Table 5.2 breaks down the market share of the top 10 most popular CAs and the number of unique binary samples signed with the code signing certificates issued by these CAs. Table 5.2 suggests that Symantec (including VeriSign and Thawte) is the most popular CA (78%) in the code signing ecosystem; 10.7% 4.1 million binary files either are self-signed or are signed by minor CAs (not included in Table 5.2). The three (*WoSign*, *Certum*, and *Startcom*) of the top 10 CAs are not by default trustful in the latest version of Windows 10.

Misuse of code signing certificates for TLS. Code signing and TLS certificates should not be used for another purpose—for example, code signing certificates cannot be used for HTTPS. We query for the keywords [6] `censys.io` [73], a web service that periodically collects TLS certificates by connecting websites. The keywords explicitly indicate that the certificates are for code signing usage. We identify 122 code signing certificates used for TLS. For example, a website, "*marketedge.com*" in the Alexa Top 1 Million domains, uses a code signing certificate for the domain without "www," but uses a TLS certificate is properly used for the domain with "www." It suggests that people including web administrators unlikely tend to differentiate between the two types of certificates.

Signed installer or application. Code signing best practices [74] recommend that installers, as well as installed files, are signed. This protects the installed files from being tampering; for example, adversaries can copy the malicious code

[6] 443.https.tls.certificate.parsed.extensions.extended_key_usage.code_signing:true

into the benign files. We examine that the best practice is well followed. The WINE data set to allow me to determine when binaries files are created on disk by their parents' process (typically downloaders or installers). We extract install and download events by employing the binary reputation dataset in WINE, which helps identify 25,051,415 unique installation and download events by counting the unique installer and downloader and payload pairs in the dataset. In these events, 2,946,950 events (11.8%) have both the signed installer and signed downloader and the signed payloads, and 666,350 events (2.66%) have installer and downloader and the payloads signed by the same publisher; meanwhile, 19,752,592 unique unsigned files were installed by the downloaders and installers.

5.3.2.3 Malformed Digital Signatures

In the data set, 101 binary samples are invalid, which means that its signature and its *authentihash* do not match. Invalid (malformed) signatures' error messages from the *sigcheck* tool is "the digital signature of the object did not verify (0x80096010)." This invalid signature results from simply copying a digital signature and a code signing certificate from a binary sample to another one. In other words, it is not related to a breach of trust in the publishers or the CAs side because adversaries do not need to have or steal a private key in this case. However, these malformed signatures account for 31.1% of the total signed malware in the data set. We are curious why many malware with malformed signatures are found in our dataset, and we conduct an experiment to determine if such digital signatures

can help the malware with malformed signatures bypass clients' protections of web browsers, AV products, and operating systems.

Browser protections. Microsoft IE9 and Google Chrome include *SmartScreen* and *Safe Browsing* respectively. It helps clients protect against malicious executable files from the Internet. To measure these protections, we conduct an experiment where we simply copy a legitimate certificate and signature to a benign, simple calculator file. The calculator file does not require elevated privileges. The copying signatures result in an executable file with a malformed signature. Then, we download the file from the Web on Chrome and IE9, as both of these browsers provide protection against malware (Google Safe Browsing and SmartScreen). Both Safe Browsing and SmartScreen blocked the sample with a malformed signature. However, we find that if the sample's extension is removed (`.exe`), it bypasses the protections.

Operating system protections. Windows platforms show a warning message to clients when the executable files come from the Web. However, if the file does not come from the Web (for example, it is copied from a USB drive), executing the file does not trigger any warning messages. We also tested another executable file with a malformed signature that needs elevated privileges. This executable triggers a warning message saying that the file comes from an unknown source when a privilege is asked to elevate regardless of where the file comes. Note that this is the same warning as the unsigned binary files. Windows can detect malformed signatures, but it does not prevent clients from executing the binaries with malformed signatures. No further warnings and checks are performed if a user ignores the warning. In short,

Windows platforms provide minimal protection for clients against executables with malformed signatures and web browsers defenses apply only to executable files from the Web; the last defense, therefore, is AV engines.

Anti-virus protections. To understand how malformed signatures can affect the AV engines, we conduct an experiment where we first download five random unsigned ransomware samples recently reported to VirusTotal. These binaries are obviously malware since they are classified as ransomware by 56-58 (more than 90%) AV engines. We extract two already-expired code signing certificates and signatures from two benign executable samples. These signatures and code signing certificates are already misused for malformed signatures in the wild. We copy the two signatures and two certificates to each ransomware sample; that is, in total there are ten samples with the malformed signatures.

Surprisingly, we find that such naive attack helps malware avoid AVs detection. Table 5.3 presents the AVs and the number of samples they failed to detect. The impact of this simple attack varies with the AV engines. The most affected AVs are *nProtect*, *Tencent*, and *Paloalto*. In particular, the AVs detect unsigned ransomware but classify eight of the ten crafted samples as benign. The malformed signatures averagely reduce the VirusTotal detection rate of r_{mal} by 20.7%. We believe that this is because AVs take digital signatures into account when filtering and prioritizing executable files to scan to reduce the overhead of inspecting the binaries on users' computers. Such the naive attack can help adversaries exploit the malformed signatures to evade AV detection. We have reported this issue to the

nProtect	8	F-Prot	4	Symantec	2	Sophos	2
Tencent	8	CrowdStrike	4	TrendMicro-HouseCall	2	SentinelOne	2
Paloalto	8	ClamAV	4	Avira	2	VBA32	2
AegisLab	7	VIPRE	4	Microsoft	2	Zillya	1
TheHacker	6	AVware	4	Fortinet	2	Qihoo-360	1
CAT-QuickHeal	6	Ikarus	4	ViRobot	2	Kaspersky	1
Comodo	6	Bkav	3	K7GW	2	ZoneAlarm	1
Rising	5	TrendMicro	3	K7AntiVirus	2		
Cyren	4	Malwarebytes	2	NANO-Antivirus	2		

Table 5.3: **Bogus Digest Detection (AV and the number of detection fail)** [2].

affected AV companies. One of them confirms that they had the issue in their AV product and plans to fix the issue. Another AV company gives another confirmation but does not provide details about this issue.

5.3.2.4 Properly Signed Malware

189 malware samples in the data set are properly signed with 111 unique code signing certificates. To sign them, adversaries must have private keys associated with the certificates. we first investigate how these code signing certificates are misused for malware in the wild and how long clients are exposed to these security threats. Of 111 code signing certificates, 27 certificates have been revoked. All executable files signed with these revoked certificates can be valid as long as they are properly trusted-timestamped. We notify the CAs of the compromised code

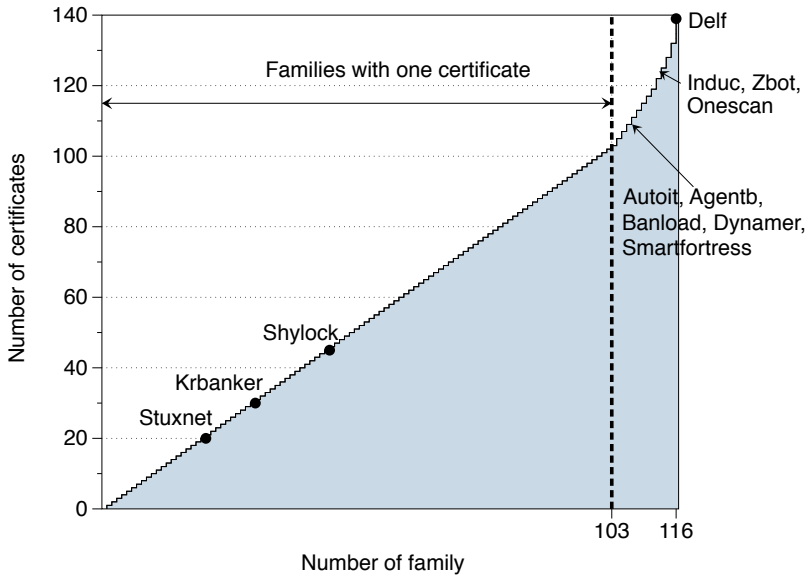

Figure 5.4: **Number of unique certificates per family** [2].

signing certificates and ask them for revocations except for two CAs (GlobalSign and Go Daddy) cause of their abuse report systems have submission errors.

Malware families. We utilize AVClass [75] to label the malware family. We identify a total of 116 malware families in the 189 properly signed malware. The most popular malware family is *delf* (seven samples), followed by *fareit* (four samples). As illustrated in Figure 5.4, 103 malware families use a single code signing certificate for each single malware, and 13 malware families use more than two code signing certificates. Of the families with multiple certificates, we observe that the malware families are droppers (*delf*, *banload*, *agentb*, *dynamer*, *autoit*), fake AVs (*onescan*, *smartfortress*), and bots (*Zeus*). However, malware families signed with each single

code signing certificate are used for the targeted attacks; e.g., Krbanker (targeted customers of South Korean banks) and Shylock (targeted customers of UK banks. Moreover, most of the signed malware (88.8%) rely on a single code signing certificate, which indicates that abusive code signing certificates are controlled and misused by malware authors than by third parties.

Certificates. An abusive code signing certificate is averagely used to sign 1.5 malware families. Most code signing certificates (79.3%) were issued to software publishers in five countries (China, Korea, USA, Brazil, and the UK). We believe that this observation reflects the software publishers' reputation used for targeted victims. In particular, targeted attacks against victims (individuals or organizations) are located in one of the countries.

TSA. Most signed malware (111, 66.8%) were trusted-timestamped; VeriSign was preferred for TSA (38, 34.2%), which suggests that malware authors consider extending the validity of their signed malware beyond its expiration dates.

Certificate lifecycle. As mentioned in Section 2.3.3, trusted timestamping can extends the validity of signed executable files even after its code signing certificate' expiration date. To understand how long clients are exposed to these security threats, we examine the abusive code signing certificates' lifecycle. We take a look at 1) the expiration date for each code signing certificate, 2) the revocation date in the CRLs (if possible), and 3) the compromised date when benign and malicious executable files are signed with the compromised certificates. If an executable file has a trusted timestamp, we consider it as a signing date. Otherwise, the earliest

date of either a first submission date in WINE or VT is considered as a signed date.

Figure 5.5 illustrates the lifecycle of abusive code signing certificates. For example, *Stuxnet* [14] is one of sophisticated attacks. The compromised code signing certificate misused for Stuxnet had been used to sign a lot of benign, legitimate executable files. the code signing certificate is revoked and the effective revocation date is set to the date when the malware is signed after the abuse is discovered. We tend to believe that Stuxnet is the earliest abuse cases, but we observe that it happened in 2003 for a code signing certificate from *Skyline Software*. Interestingly, we find that five code signing certificates were misused to sign malware, not trusted timestamped, but they are found after its certificate expiration date. Adversaries have no motivation, but an attempt to evade AV detection as discussed in Section 5.3.2.3. We believe that the attack has been present in the wild for a relatively long time.

To understand how long the compromised code signing certificate remains a security threat, we utilize the survival analysis [76], which can estimate the probability of the survival of the compromised code signing certificates (i.e., they are not revoked). A signing date of the oldest malware sample signed with a certain code signing certificate is considered as the *birthday* of the abuse. We estimate *death events* (when the certificates are revoked). For a revoked certificate, we utilize the scanning date of VirusTotal reports, and consider the most last date when the state is "valid." Then, we compute the time difference in days between the revocation date and the timestamping date or the first appearance date of the oldest malware signed by the same certificate. This is a conservative estimation of how long the security threats are exposed since the birthday is an upper bound for when a cer-

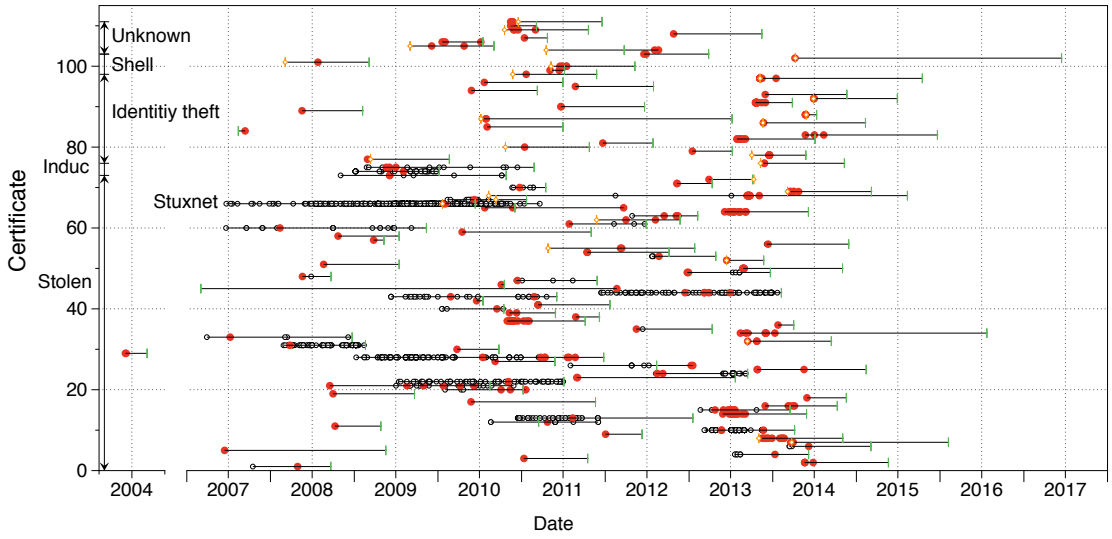

Figure 5.5: **Lifecycle of the abusive certificates** [2]

tificate becomes compromised and the death date is a lower bound for when the certificate revoked.

We calculate the Kaplan-Meier estimator [76] for the estimation of the survival as presented in Figure 5.6. 96% compromised code signing certificate can survive after the first date. The probability keeps to decrease continuously, but slowly for 5.6 years. Then it stops at 80%. This indicates that security threats are resilient. Only 20% code signing certificates misused to signed malware are likely to be revoked. Even worse, if the malware is signed and trusted-timestamped, it can be valid even today.

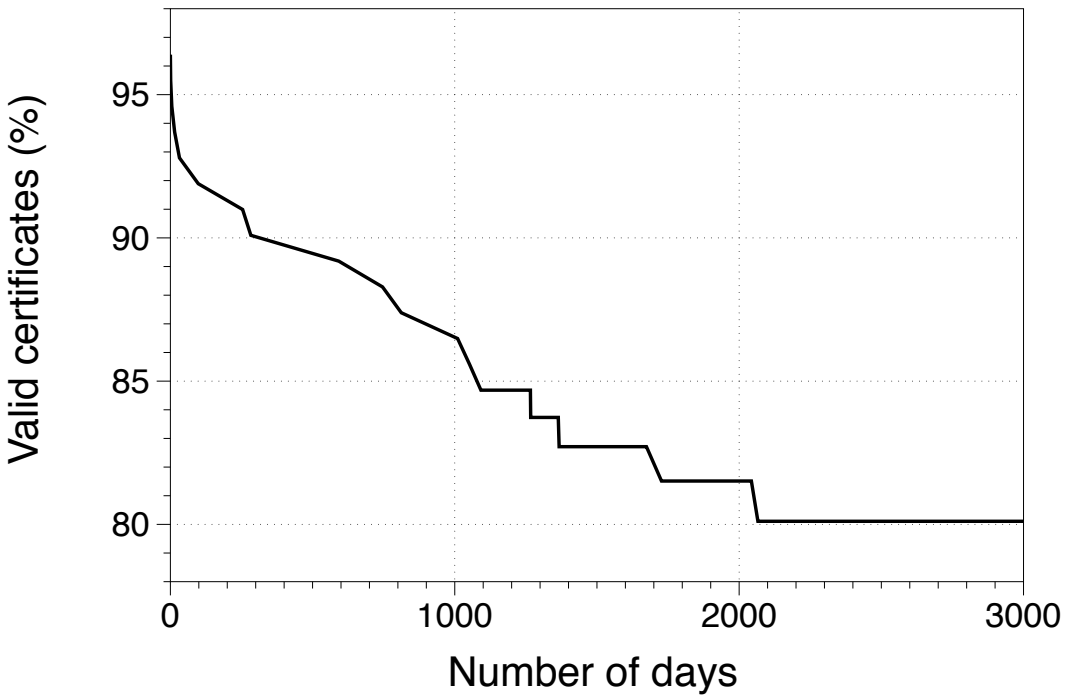

Figure 5.6: **Estimation of the threat effectiveness** [2]

5.3.2.5 Measuring the Abuse Factors

We more deeply examine the properly signed malware to answer our research question of "How and why code signing abuse happened," using the abuse detection algorithm from Section 5.3.1.3.

Publisher-side key mismanagement. A code signing certificate is considered being resulted from this category if the benign and malicious executable files are found in the same group of a certain certificate. Of the 111 groups, 75 certificates are

Compromised		Identify Theft		Shell Company	
Issuer	Count	Issuer	Count	Issuer	Count
Thawte	27	Thawte	8	Wosign	2
VeriSign	24	Comodo	4	DigiCert	1
Comodo	8	VeriSign	4	USERTrust	1
USERTrust	2	eBiz Networks	3	GlobalSign	1
Certum	2	USERTrust	1		
Others	9	Others	2		
Total	72 (64.9%)	Total	22 (19.8%)	Total	5 (4.5%)

Table 5.4: **Type of abuse and the top 5 frequent CAs** [2]

used to signing both malicious and benign samples. Surprisingly, most (50, 66.7%) certificates are still valid; in other words, they are not revoked yet. To categorize the compromised code signing certificate, we manually investigate malware samples signed with the certificates.

- ***Compromised certificate.*** Of 75 code signing certificates, we believe that most (72) were compromised and misused for signing malware. In other words, the private keys associated with the certificates are stolen and misused to sign malware. In the data set, we find that *Stuxnet* malware [14] signed with the *Realtek Semiconductor Corp.* certificate issued from *VeriSign*. Our system also detects that an Australian department's private key was also stolen and misused to sign *autoit* malware.

- ***Infected developer machines.*** We also identify infected developer machines that affect all benign executable files built on the developer machines. It automatically includes malicious payloads into benign samples and signs them

with a legitimate code signing certificate, which results in signed malicious executables distributed with a legitimate package. We find that three code singing certificates misused to sign *W32/Induc.A* malware that infects only *Delphi* developer machines. We also investigate the prevalence of *W32/Induc.A* malware in the wild. Approximately 1,554 executable files are detected as *W32/Induc.A* and 93,016 machines were infected. Among these machines, the only 180 were Delphi compiler machines used for developers. This indicates that infecting developer machines is a very efficient way to amplify the impact of signed malware and infect 517× more machines.

Table 5.4 depicts that 70% of the code signing certificates in this category are issued by Symantec group (*VeriSign* and *Thawte*).

CA-side verification failure. The weakness in this category results from CAs' failure in their vetting process when issuing code signing certificates. For example, a CA may issue a code signing certificate to a malware author who uses fake information by 1) impersonating others 2) establishing shell companies. We believe that 27 code signing certificates are is-issued to adversaries because of verification failures. We also manually investigate each code signing certificate by searching the company name on the Internet or in *openCorporates* to check if the companies are legitimate and to distinguish between shell companies or identity theft. 22 code signing certificates (out of 27 certificates) are mis-issued through identify theft, and 5 are done through shell company.

To understand the weakness in this category, we also investigate the vetting

process before issuing code signing certificates. At the end of 2016, Certificate Authority Security Council (CASC) announced a minimum requirement for code signing certificates [38]. Microsoft announced the CAs must follow the minimum requirements after February 1, 2017. The new requirements include:

- ***Stronger protection of private keys***. Private keys must be stored on secure cryptographic hardware—e.g., a USB token or Hardware security module (HSM).
- ***Careful identity verification***. Strictly verification for the identity of the publisher is required, which also includes the cross-checking with the blacklist for bad publishers.
- ***Better response to the abuse***. Quick responses to the revocation requests are required within two days.
- ***TSA is now a requirement***. Every CAs must operate an RFC-3161 compliant TSA.

Verification and further investigation. We contact the owners of the compromised code signing certificate found in this work, 1) to inform them that their code signing certificates are compromised and abused, and 2) to better understand the code signing ecosystem. We manually look for their information on the Internet and send emails to 23 publishers to ask them if they have owned the code signing certificates and are aware of this issue. We are not able to send emails to more publishers since their information is not searchable on the Internet. We receive eight emails

from them. All of them said that they had the code signing certificates and used to sign only their benign programs. Three of them already recognized that their code signing certificates were abused and revoked because they were notified that their code signing certificates were compromised. A publisher said that their private key might be stolen because they shared with other people the machine where their private key was stored.

5.4 Code Signing Certificates Traded in Black Markets

The first goal of malware authors is to conceal their identities on their signed malware samples. However, it is very difficult for them to legitimately obtain code signing certificates from CAs. This is because the current CAs' vetting processes prevent them from obtaining code signing certificates to sign their malware unless they provide their identities to the CAs. Therefore, as discussed in the previous Section 5.3, the malware authors establish shell companies or steal others' identities to sign their malware, which raises research questions "*how do malware authors legitimately acquire code signing certificates?*" and "*is there another channel where malware authors readily obtain code signing certificates?*"

Recent anecdotal reports indicate that on underground markets code signing certificates are also traded [77–79]. These underground markets allow malware authors to readily purchase a new code signing certificate with a new publisher identity to use it for signing their malware. However, these anecdotal reports regarding the underground economy are not well understood. In particular, the economic driving

for code signing certificates trades in underground markets has not been analyzed yet.

In this work[7], we measure and analyze the underground trade for code signing certificates with the whole ecosystems of vendors, malware authors, certificates issuers.

5.4.1 Data Collection

To answer the research questions, we first crawl the black market for code signing certificates and passively collect black market data[8]. We analyze black market vendors and their business models and observe their activities in underground markets. However, there are challenges in data collection about black markets for code signing certificates. Particularly, there is no easy way to locate the black markets where code signing certificates are traded. The prior report described an online black market for code signing certificates [77], but the online marketplace is no longer active as of this work. Moreover, Christin collected *SilkRoad*, black market in darkweb and released the data sets [80]. However, our interest in code signing certificates is not found in the released data set; rather SilkRoad is a general marketplace such as narcotics.

To overcome this challenge, we begin with a handful number of well-known websites (such as hacking forums and marketplaces). We utilize these websites to

[7]Published as [3]

[8]We do not interact with any black market vendors such purchasing code signing certificates or exchange messages.

expand the data sets for black markets. First we start an investigation on the black market vendors who sell code signing certificates. Then, we analyze information posted on the websites over time: for example, how big the business is, how much profit they make, etc. These works are manually conducted because of 1) a large variety of goods and 2) anti-crawling protections which prevent me from automatic crawling the websites.

Forums and Marketplaces. We start with a small number of well-known hacking forums (e.g., *Hackforums*), general marketplaces on darkweb (e.g., *Dream Market*), and link directory sites for darkweb (e.g., *Torlinks*). Then, we search some keywords related to code signing certificates (such as "code signing," "certificates," etc.) on these websites. We keep expanding the data collection until new closed websites are found that we cannot access.

In total, we found 28 forums, 6 link directory websites for darkweb, and 4 general marketplaces. Of these 4 marketplaces, only one website sells only code signing certificates, called *CodeSigning Guru*.

Vendors and Purchases. We find in total four vendors who sell code signing certificates in multiple forums (with the same user ID or advertisement), and one e-shop, called CodeSigning Guru, launched and operated by one of the four vendors. We first collect information regarding the vendor activities such as registration date, last edit date, post date, etc.). In particular, our focus is on stock updates and

Vendor	Activity & Presence			Inferred Sales Volume			Products	
	First Joined	Last Reply	Forums	Vouches	Updates	E-Shop	Item: pieces	Price ($)
A	2011-09-07	2015-05-25	3	0	-	-	Standard: 1 Standard: 10+ Standard: 15+	1000 800 700
B (Codesigning Guru)	2016-01-08	2017-08-08	8	3	7+	41	Standard (Comodo) Standard (Thawte) EV	350 500 2500
C	2016-09-09	2017-01-27	10	0	-	-	EV (earlier posts) EV (current)	1600 3000
D	2017-03-07	2017-08-13	2	4	7+	-	Standard: 1 Standard: 5 Standard: 10 w/ SmartScreeen rep.: 1 w/ SmartScreeen rep.: 5 w/ SmartScreeen rep.: 10	400 1700 3500 800 3700 7000

Table 5.5: **The leading black market vendors** [3]

vouches. This information of the remaining stock was posted by the vendor. The payments for CodeSigning Guru e-shop are made through Selly[9]. Selly is a web platform where vendors can sell their items by hiding their identities and clients can buy the items vice versa. The payment will be made typically using cryptocurrency such as *Bitcoin*. Therefore, we implement a crawler that collects the remaining stock every five minutes from Aug. 25th, 2017 and Dec. 7th, 2017. This remaining stock information helps to understand the size of the black market business.

5.4.2 Code Signing Certificate Black Market

Black markets are dominated by four vendors (A-D). we manually investigate these four vendors and Table 5.5 summarizes the vendor business activities such as the periods of activity, sizes of their business, and the prices of code signing

[9]https://selly.gg/

certificates.

5.4.2.1 Vendors and Activity

Vendor D operates on two English speaking forums, and vendors A and C do on 10 and 3 Russian speaking forums respectively, but vendor B operates on both three Russian and five English speaking forums. Moreover, vendor B extends her/his business by starting up a new e-shop for code signing certificates, called *CodeSigning Guru* in August 2017. Good and prices are the same as the posts that they made on hacking forums. The vendor also advertises the e-shop on the forums for potential customers.

The oldest vendor A is not very active now, but he/she had posted his/her selling code signing certificates from 2015 with no more updates. However, vendor B, C, and D more than half of their posts are made after May 2017. The two most active vendors B and D regularly update their posts around once or twice a month and reply to any questions to the original posts such as remaining stocks.

They typically start a new thread on a forum and update the thread over time with price changes, new features, etc. All transactions are made in private; typically over instances messaging services such as telegrams or jabber. Therefore, it is very difficult to know the estimation of sales. They keep using the same user IDs for their reputation, and the same content of the posts.

The previous report [77] presented Signature-as-a-Service (**SaaS**) in the past. However, in our measurement, we are unable to find any evidence of the SaaS

business model. Rather they sell code signing certificates and malware authors purchase the certificates to sign their malware themselves.

5.4.2.2 Goods and Deals

Vendors A and D sell standard code signing certificates, but vendor C offers only EV code signing certificates. Vendor B who operates *CodeSigning Guru* provides both standard and EV code signing certificates. All of them claim that their certificates are very fresh; in other words, they are directly issued from CAs. Some vendors ask customers (malware authors) to pay the half of the price since code signing certificates are issued from CAs on demand. Vendor A, interestingly claims that he/she always has a few of publisher identities to apply for a new code signing certificate; in turn, customers (malware authors) can choose one of the publisher identities.

Vendors also explain what the code signing is for beginners. Particularly, they explicitly explain that standard code signing certificates can be considered as non-benign ones. This is because code signing certificates have to build their own reputation. Vendors provide details about how to bypass *SmartScreen* by building reputations for code signing certificates. Interestingly, vendor D offers code signing certificates with an already-built reputation. Vendor B mentions on the FAQ of his/her e-shop website, to build a reputation, around 2,000 – 3,000 installations of benign files are required on Windows 10 systems, which helps avoid warning messages when the signed files are executed on Windows platforms.

Certificates	Total		Average per month	
	Items sold	Revenue ($)	Items sold	Revenue ($)
Comodo	29	10,150	8.36	2,928
Thawte	12	6,000	3.46	1,731
Total	41	16,150	11.83	4,659

Table 5.6: **Sales volume recorded on Codesigning Guru** [3]

Prices for standard code signing certificates range between $350 to $500. Vendor D sells code signing certificates issued from Comodo, Thawte, DigiCert, and Symantec in $400. However, Vendor B sells code signing certificates at different prices; for example, a certificate from Comodo costs at $350, a Thawte one does at $500. He/she claims that a Thawte code signing certificate has more trustful. EV certificates that do not require reputation building are more expensive. In the earliest post, vendor C sells an EV certificate at $1,600, and the most recent posts, the same vendor does an EV certificate at $3,000, which means the price increases. Meanwhile, vendor B sells an EV certificate at $2,500 both from the forums and the e-shop website. The EV certificate requires a USB token for storing the associated private key. The USB will be sent to customers by vendors.

5.4.2.3 Sales Volume

All transactions on forums are hard to know or estimate since they made in private. As shown in Table 5.5, we report the number of vouches for each vendor. Vouches are the way to establish trust in the black market by leaving comments

or reviews of purchased code signing certificates. However, not all customers write comments or reviews so that these vouches cannot be feasible to estimate the sales volume.

The most active vendor B regularly updates their remaining stock on their e-shop website. The stock updates can be used to estimate the vendor B's sales volume and their profits. The e-shop sells standard code signing certificates from Comodo or Thawte, and EV certificates from unspecified a CA. During the observation period (Aug. 25th, 2017 – Dec. 7th, 2017), we observe that 41 standard certificates are sold. On average, 11.8 code signing certificates are sold per month, which brings in a total of $16,150 revenue (refer to Table 5.6).

5.5 Revocation Effectiveness in Code Signing PKI

Code signing certificates can be compromised due to several reasons as discussed in Section 5.3, or they are mis-issued directly to malware authors or black market vendors as presented in Section 5.4. In Section 5.3, we find that malware authors stole the private keys of legitimate code signing certificates that have been used to sign benign executable files. Then, the stolen private keys are misused to sign their malicious executable files. As discussed in Section 5.4, malware authors or black market vendors obtain standard and EV code signing certificates legitimately from CAs. They also misuse the mis-issued code signing certificate for their malicious executable files.

The primary defense against the abuse is revocation. In other words, the

compromised or misused code signing certificates must be revoked by the issuing CAs. In the Web PKI, prior works have revealed significant problems including long revocation delays [27,43,44], heavy bandwidth costs for dissemination [46], and clients misbehavior (e.g., web browsers do not properly check revocation status) [46]. By contrast, little is known about the revocation of the Code-Signing PKI. The scarce understanding can mislead platform security protections to having incorrect assumptions when handling code signing certificates.

Moreover, the validity of code signing certificates can be extended as long as a binary sample is trusted timestamped. If a binary sample is properly signed and trusted timestamped, the binary sample can be valid after its certificate expiration date, which remains security threats if it is malware. Therefore, the code signing certificates should be revoked even after expired. This is a critical issue in the code signing PKI.

In this work[10], we measure and analyze the effectiveness of revocation in the code signing PKI. First, we identify the effective revocation process; 1) promptly discover misused code signing certificates, 2) properly revoke the certificates, and 3) properly disseminate the revocation information to clients. Moreover, we find out what security threats can be introduced if the revocation process is not effective.

Unlike the prior studies in the Web PKI [41,44–46,81] where TLS certificates can be collected by scanning the Internet, we are not able to utilize a comprehensive corpus of code signing certificates because there is no official repository for code signing certificates. To overcome the challenge, we use data sets that are publicly

[10]Published as [4].

released from prior research [12, 13] and use Symantec's internal repository of binary samples to increase our coverage. We extract 145,582 distinct leaf code signing certificates from the dataset. From the code signing certificates, we also extract 215 distinct Certificate Revocation Lists (CRLs) used only for code signing certificates, and distinct 131 Online Certificate Status Protocol (OCSP) points. We periodically probe the collected CRLs to check their revocation status to collect the revocation publication date when a certificate is revoked by a CA and the revocation information is disseminated.

In short, we make the following contributions:

1. We collected a large corpus of code signing certificates as well as revocation information of the certificates.

2. We conducted end-to-end measurement of the revocation process in the code signing PKI.

3. We estimated a lower bound on the number of compromised certificates.

4. We highlighted the security problems in the three steps in the revocation process.

5.5.1 Discovery of Potentially Compromised Certificates

Due to many reasons, code signing certificates must be revoked. Generally, it is hard to determine code signing certificates should be revoked or when they should be revoked. However, in one situation we can be certain that prompt certificate

revocation is necessary; when the corresponding private keys have been compromised or are in the hands of malicious actors to use to sign their malware [38].

Therefore, we estimate a conservative number of compromised code signing certificate used to sign malware in the wild. Moreover, we compare the estimated number with the coverage of a major security company to better understand how difficult to discover all potentially compromised certificates in the wild (Section 5.5.1.1). Furthermore, CAs must take action of revoking the compromised certificates after signed malware has been discovered, and they add the revoked certificates' serial numbers to CRLs. We measure the time delays between 1) the date when signed malware appears in the wild and 2) the date when the revoked certificates are added to CRLs (Section 5.5.1.2).

5.5.1.1 Mark-recapture Population Estimation

The revocation process begins by discovering the compromised certificates misused to sign malware. To measure how effectively CAs discover malware samples signed with their code signing certificates, we employ the mark-recapture analysis [82] because there is no official repository for code signing certificates and signed executable files.

The mark-recapture analysis is to estimate the size of N of a wildlife population not observed in the entirety. Two separate samples are required. 1) The first samples should be the random capture of n_1 subjects; then the samples are marked and released in the wild. 2) The second samples are the random capture of n_2

subjects among p are the markings from the first sampling. In other words, p is the intersection of the first and second samples. An estimated $\hat{N}$ for the total population of N can be calculated as:

$$\hat{N} = \frac{n_1 n_2}{p} \tag{5.1}$$

For this study, N becomes the total number of code signing certificates misused for signed malware. To apply this technique to this study, we utilize two different data sets: VirusTotal and Symantec telemetry. We consider each data set is samples of potentially compromised code signing certificates. In particular, n_1 and n_2 represent the numbers of certificates to be revoked since they are misused to sign malware from VirusTotal and Symantec data sets respectively.

Assumptions and interpretation. The Mark-recapture technique requires three assumptions regarding the population and the sampling process, but the sampling process does not fit this study. First, the subjects of the population must be captured with an equal chance, which means the population is *homogeneous*. However, the population of code signing certificates is unlikely homogeneous. For instance, a popular software company's code signing certificate may more likely appear in these two data sets. Second, the samples have to be *independent*; in other words, the first sampling (capturing) should not affect the second recapturing.

To mitigate the first issue (homogeneousness), we first compute the daily es-

timates between Apr. 18, 2017 and Sept. 10, 2017. Some certificates issued before Apr. 2017 can be included in the two data sets. However, we believe that malware signed with the older code signing certificates may not have a high chance of appearing in the VirusTotal Hunting. Moreover, some other code signing certificates are misused for only targeted attacks; accordingly, the code signing certificates may nor appear in either the two data sets.

To minimize the second issue (Independence), we also estimate $\hat{N}$ separately for each day and consider the birth date for each code signing certificate as the first seems to timestamp in Symantec telemetry and VirusTotal. Furthermore, we consider the revocation date (t_p)as the death date of a code signing certificate; in other words, the revoked code signing certificate leaves the population. The population is closed within each day because CRLs are collected and updated every day.

Note that these approaches for the estimated population ($\hat{N}$) would underestimate the real entire population of potentially compromised code signing certificates (N). Similarly, the intersections (p) between the two data sets will increase when the dependencies are large, which leads to an underestimation of N. In short, this estimation in this work must be interpreted as a *lower bound* for the real total population of potentially compromised certificates.

Results. Taking into account the considerations listed above, we estimate a lower bound number of signing certificates used to sign malicious binaries in the wild. The

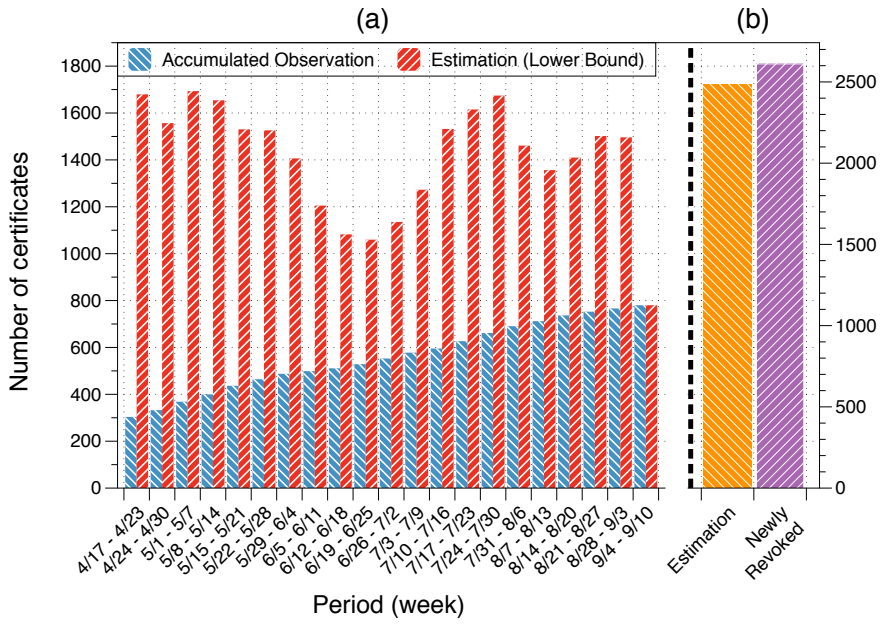

Figure 5.7: **Mark-recapture estimation** [4].

estimation is done per day starting from the birth of the oldest certificate in the dataset.

Figure 5.7 shows two graphs: a) trend in malware signing certificates (mark-recapture estimation as red and observed number as blue) over time and b) comparison between the estimation and the total number of newly revoked certificates during (4/18/17 – 9/10/17) (the label starts from 4/17 since it is the start of that week. Particularly, a) shows the average of the daily estimations $\hat{N}$ for each week during our observation period. We also compare the estimation with the actual number of the potentially compromised certificate that is the union of the two data sets of Symantec and VirusTotal.

We exclude the last week (9/4 – 9/10) because the estimation and the observation are identical due to the fact that eventually most of the certificates got revoked for the ones from the Symantec telemetry. We estimate that at least 1,004 – 1,786 code signing certificates were misused for malware in the wild. The misused certificates are not revoked by the date of the estimation. The estimated population is averagely 2.74× bigger than the actual observed number of certificates. It indicates that even the biggest malware collector, VirusTotal and a major security company, Symantec are unable to capture all real population of the potentially compromised certificates.

5.5.1.2 Revocation Delay

According to the minimum requirement [38], CAs must revoke a code signing certificate within seven days after they discover the certificate becomes compromised and misused for malware. As described in Section 5.3, the compromised code signing certificates are estimated to remain valid for more than 5.6 years after they are first misused to sign malware. However, it is estimation so that in this section, we measure how promptly CAs revoke their compromised certificates using a data-driven approach. This measurement requires an accurate estimate of a revocation publication date (t_p). Therefore, we develop a crawler that collects revocation publication dates between Apr. 16th, 2017 and Sept. 10th, 2017.

Another challenge we have is to know when signed malware appears in the wild and they are discovered by CAs. We use Symantec metadata telemetry (see

Section 5.2.3) to know executable files signed with revoked code signing certificates. Of 2,617 revoked code signing certificates, 468 (17.9%) are found in our dataset, and 146,286 executable files signed with the revoked certificates. Because Symantec does not collect these actual executable files, we utilize VirusTotal and AVClass [75] to obtain reports of these executable files. We also use the first submission date of each executable files from VirusTotal reports. In total, 19,053 unique samples in VirusTotal are found, and 254 unique certificates are used to sign the samples.

For each code signing certificate, we use the earliest first submission date of VirusTotal as the discovery date (t_d) because we believe that AV companies and CAs should monitor VirusTotal and check if their code signing certificates are misused to sign malware. We compute the *revocation delay* ($t_p - t_d$) as the difference between the earliest scanning date of malware in each group (t_d) and the date when revoked and disseminated (*revocation publication date* (t_p)).

Results. Figure 5.8 shows a cumulative distribution of the revocation delay between the dates on which the malware signed with compromised certificates and the dates on which CAs revoke the compromised certificate. The revocation delay is between one day and 1,553 days. The average delay is 171.4 days (5.6 months) (std 324.9 days, median 38 days) to revoke their certificates after the malware signed with the compromised certificates appear in the wild. The delay indicates that CAs do not always strictly follow the requirements yet, or there is not enough coordination to get early alerts on newly discovered signed malware. Consequently, Windows clients remain exposed to this security threat for over five months averagely after signed

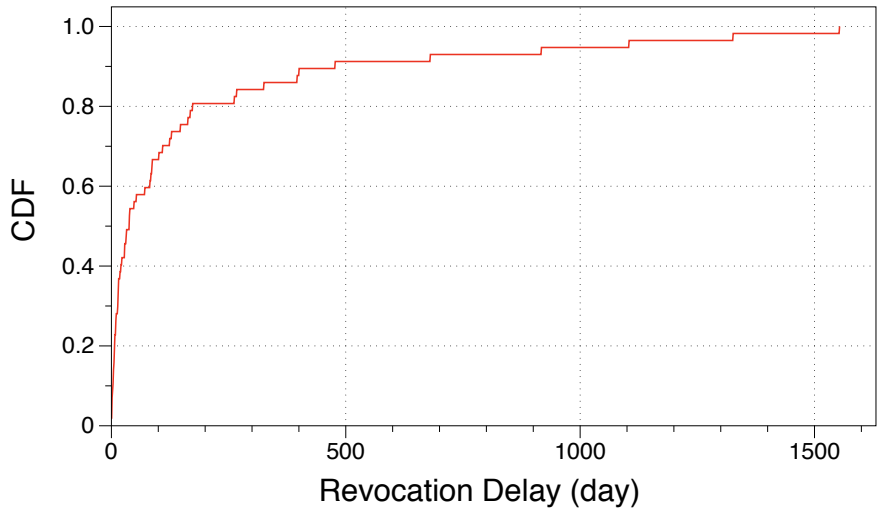

Figure 5.8: **Revocation delays.** The delay is between the date when the malware is signed with an abusive code signing certificate and the date when the issuing CA revoke the compromised certificate. [4]

malware appears in the wild.

5.5.2 Setting the Revocation Date

To make all signed malicious executable files invalid, CAs must determine a proper revocation date, called *effective revocation date* after potentially compromised code signing certificates are discovered. In this section, we measure how properly CAs set effective revocation dates.

5.5.2.1 Problems in Revocation Date Setting

As described in Section 2.3.4, CAs must set a proper effective revocation date when revoking compromised code signing certificates. There are the two ways of setting effective revocation dates; 1) hard revocation and 2) soft revocation. In hard revocation, CAs set t_r (effective revocation date) to t_i (issue date), which leads to software publishers re-signing and re-distributing their software. On the contrast, when t_r is set between t_i (issue date) and t_e (expiration date), it is called soft revocation. In soft revocation, if an effective revocation date is properly determined and set, it would be a perfect solution. This is because software publishers do not need to re-sign and re-distribute their software while all malicious executable files signed with the compromised certificates become invalid. We examine what revocation date setting policies CAs apply and how the policy trends have been changed over time. We also identify security threats caused by the erroneous effective revocation date setting in soft revocation.

Trend of effective revocation date setting. We measure what revocation date setting policy CAs to apply when revoking compromised certificates using the collected 145,582 code signing certificates (Section 5.2.3). We first utilize CRLs for revocation status specified at its x509 v3 code signing certificate extension field.

Table 5.7 breaks down the effective revocation date setting policy. We find that 5,410 (3.7% out of 145,582) code signing certificates have been explicitly revoked. Of those, 96% (5,196) code signing certificates were issued by the top 10 CAs. Most (1,880, 34.8%) revoked code signing certificates were issued by *Comodo*, followed by

	$< t_i$	$= t_i$	$\leq t_e$	$> t_e$	Total
Comodo	0	426	1,437	17	1,880
Thawte	0	74	1,055	39	1,168
Go Daddy	2	14	672	18	706
Verisign	2	59	430	51	542
Digicert	1	161	323	3	488
Starfield	0	3	153	2	158
Symantec	0	33	89	1	123
Wosign	0	57	17	0	74
Startcom	0	0	47	0	47
Certum	0	1	9	0	10
Other	0	96	117	1	214
Total	5	924	4,349	132	5,410

Table 5.7: **Effective revocation date setting policy for top 10 CAs.** (t_i: issue date and t_e: expiration date) [4]

Thawte (1,168, 21.6%). Most (4,481, 82.8%) code signing certificates were revoked using the soft revocation policy while only 17.2% code signing certificates apply the hard revocation policy.

Most CAs perform both soft revocation and hard revocation. Soft revocation is more preferred than hard revocation in all CAs except for *WoSign*. Particularly, *Startcom* has never applied hard revocation for their revoked code signing certificates. Interestingly, three CAs (*Go Daddy*, *Verisign*, and *Digicert*) set the effective revocation dates for their revoked code signing certificates to before the issue dates.

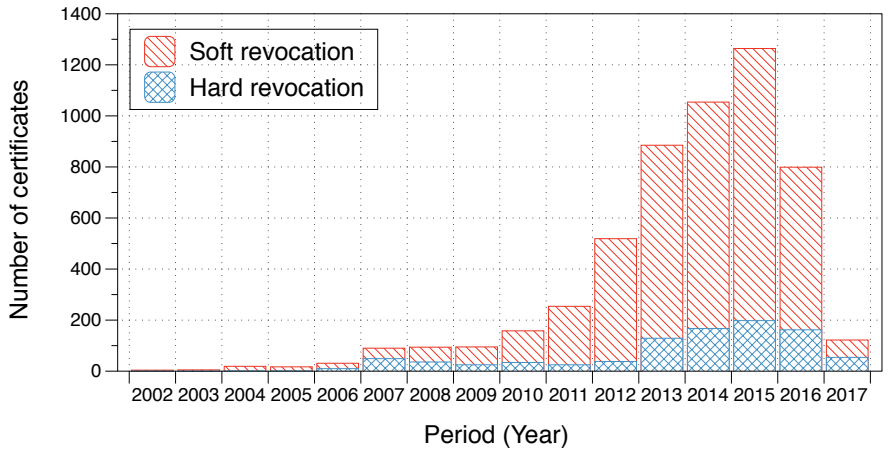

Figure 5.9: **Effective revocation date setting trends.** Number of revoked certificates (stacked). [4]

This can be considered hard revocation; thus there are no security threats for clients. Figure 5.9 shows the total number of hard and soft revocations; the total number has drastically increased since 2012. It is worth noting that the revocation numbers for 2016 and 2017 have not finally made yet. Therefore, we believe that the numbers will continue to increase in the future due to the revocation delays.

Ineffective revocation date setting. As described in Section 2.3.4, soft revocation can lead to signed malware remain valid even after the code signing certificate was revoked if the CA erroneously set effective revocation dates. As presented in Table 5.7, most CAs have experienced setting the effective revocation dates to even after its code signing certificate expiration dates. This means that the revoked code

signing certificates become ineffective so that all executable files (including malware) signed with the code signing certificates still remain valid.

We conduct a measurement study of how many CAs make mistakes by setting effective revocation dates, and accordingly, how many signed malware can be still valid due to the errors. To examine the wrong effective revocation dates, the signing date information of signed executable files is required. We utilize the WINE data set (see Section 5.2.1) and query VirusTotal with 12,351,946 signed executable files from the WINE data set. Of these, 38.3% (4,729,023) samples signed with 45,613 leaf unique certificates are found in VirusTotal. We are unable to directly query VirusTotal for all old samples and to obtain the effective revocation dates of the 45,613 code signing certificates because 1) the search index service of VirusTotal supports for only recent 80TB data or approximately one month of samples, and 2) VirusTotal does not provide any information related to revocation such as CRLs and OCSP points. Thus, we query the CRLs we have collected from Section 5.2.3 to check if a code signing certificate is revoked and to obtain an effective revocation date (t_r). This process returns 1,022 revoked certificates (out of 45,613).

Ineffective revocation date setting. We observe that CAs who applied the soft revocation policy erroneously set the effective revocation date (t_r) of 45 (5.1% out of 891) TLS certificates. We also measure how many signed malware is still valid due to the erroneous effective revocation date. We first utilize AVClass [75] to label signed malware using VirusTotal reports. Among signed malware samples, if signed dates (t_m) is earlier than the effective revocation date (t_e), they are erroneously set

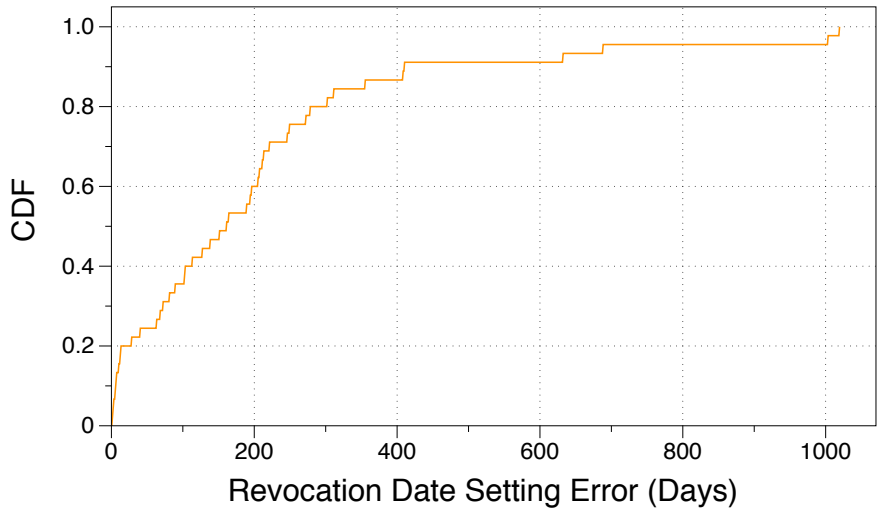

Figure 5.10: **CDF of the revocation date setting error ($t_r - t_m$).** [4]

the effective revocation date. We observe that 250 signed malware (5.3% out of 4,716 signed malware) still remain valid due to the wrong effective revocation dates, which means the validity of signed malware can result in security threats to clients. Figure 5.10 presents the difference between the oldest signing dates of the signed malware (t_m) and the effective revocation dates (t_r). The longest difference is 1,019 days (2.8 years) and the shortest one is one day. Windows clients may be exposed to the security threats where they may execute or install the still-valid signed malware because the signed malware is still valid although the corresponding code signing certificate is already revoked.

5.5.3 Dissemination of Revocation Information

The next step for CAs is to disseminate revocation status information to clients after compromised and mis-issued certificates are properly revoked and effective revocation date are correctly decided. We first look at the enforcement of the Windows platforms because the enforcement policies of checking revocation status information in client-side platforms can affect clients in terms of security.

5.5.3.1 Enforcement in Windows

Windows platforms must check the validity of code signing certificates upon encountered signed executable files. Depending on the Windows' policy that handles the failures in checking revocation status, Windows' clients will be affected when executing binary files.

The soft-fail policy is applied on the Windows' platform, which means the platforms presumably believe code signing certificates are valid even though the revocation status information is not available. Consequently, when the network or CAs' infrastructures have issues, clients are able to execute a signed binary with revoked certificates without any warnings. We find that the problems in the CAs' dissemination of revocation status information, with the soft-fail policy of the Windows platforms, can allow executable files signed with revoked certificates to be executed without security warning messages.

5.5.3.2 Unavailable Revocation Information

CAs must maintain their certificates' revocation status information for clients. More importantly, the revocation status information must be kept and disseminated indefinitely because of the trusted timestamps that extend the validity of signed executable files beyond the certificates' expiration dates. In other words, the revocation status information should be maintained and always-available longer than the certificates' life [39]. This is the most significant difference from the Web's PKI (TLS).

Certificates without CRL URLs and OCSP points. The first problem we find is that code signing certificates used to sign executable files do not include CRL URLs and OCSP points. Code signing certificates follow the x.509 v3 standard format, which means that the certificates must include CRL URLs and OCSP points for clients. In turn, clients are able to check the revocation status of code signing certificates using CRLs or OCSP.

However, we observe that 788 (0.5% out of 145,582 in Section 5.2.3) certificates do contain neither CRLs and OCSP points. This means that clients are unable to check the revocation status of these code signing certificates. Of the 788 code signing certificates, most (676, 85.8%) were issued before 2003 by *Thawte*. The most recently issued code signing certificate found in the 788 certificates it that *iTrusChina*'s one. Therefore, we can conclude that the security problem exists and persists.

These 788 code signing certificates have already expired; thus, no new executable files are unable to be signed with these code signing certificates. However, previous executable files (including malware) signed with these 788 code signing

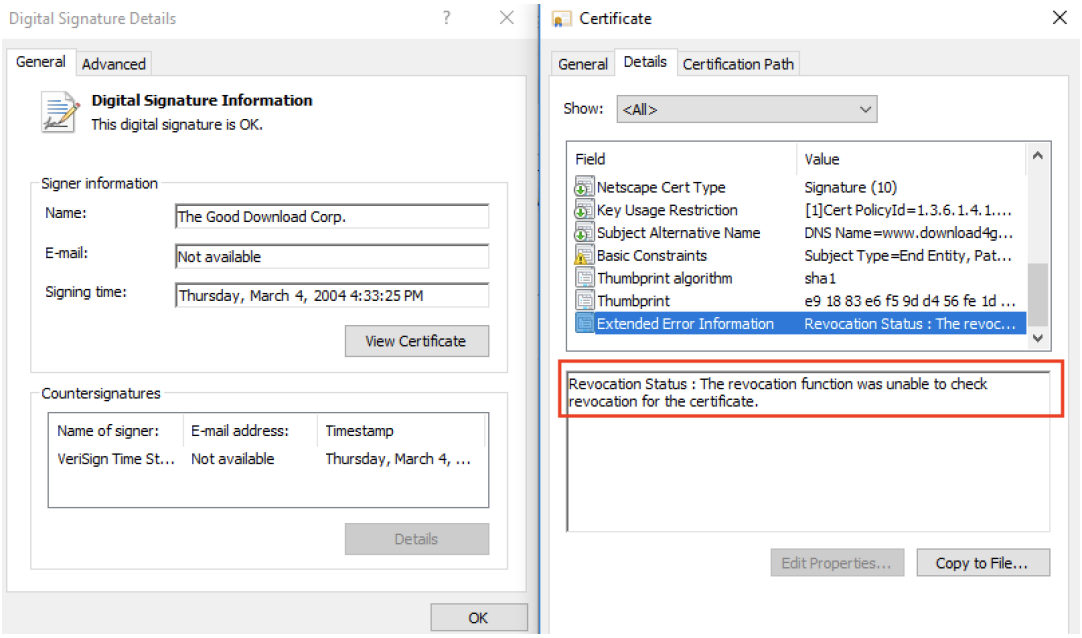

Figure 5.11: **Screenshot of Windows 10.**

certificates can be valid if they are trusted-timestamped. In other words, clients have exposed this security threat even though code signing certificates already had been revoked.

We also examine how this security threat affects Windows platforms. We obtain the executable binary samples signed with one of these 788 code signing certificates from VirusTotal. We then inspect the code signing certificate of these samples on Windows 7 and 10. We observe that on both the platforms, a message of "The revocation function was unable to check revocation for the certificate" is displayed and the code signing certificate is valid since the Windows platforms apply the *soft-fail* policy as shown in Figure 5.11 and Figure 5.12.

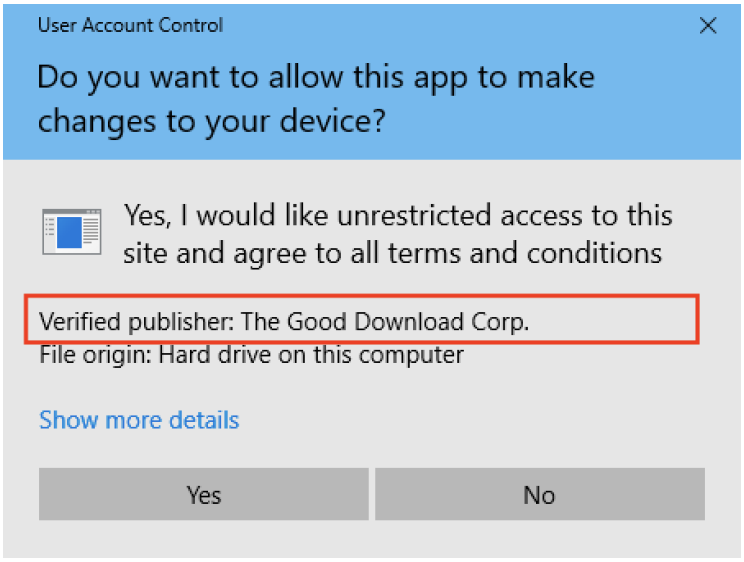

Figure 5.12: **Screenshot of the prompt displayed in Windows 10.**

Unreachable CRLs and OCSP servers. We examine the reachability of CRLs and OCSP servers specified in code signing certificates in our date set Section 5.2.3. In our observation period (Apr.16th, 2017 – Sept. 10th, 2017), We find that 55 CRLs (out of 413) are not reachable at least once a day. However, our institution had network issues that may have resulted in network failures in our reachability checking system. Therefore, we manually double-check if the 55 CRLs were typically reachable, and remove them, which remains 13 CRLs. The 13 CRLs have been never reachable during the observation period. Of the 13 CRLs, 5 (38.4%) had HTTP 404 Not Found errors; for instance, two CRLs[11][12] had the 404 not found errors. This means that the CAs have removed the CRL files from the URL locations and no

[11] http://crl.globalsign.net/ObjectSign.crl

[12] http://www.startssl.com/crtc2-crl.crl

longer update the CRL files, but the domains still exist and are being used.

Interestingly, a domain for CRLs has been obtained by a domain re-seller, and the domain is being sold now. Using *Internet Archive* (https://archive.org/) to check the history of the domain, the last time the old website was captured was in August of 2015, and the domain re-seller page appeared in September of 2015. We believe that the domain was taken between August 1st, 2015 and September 28th, 2015 because the legitimate website was last seen on Aug. 1st, 2015 and the domain re-seller's webpage was seen on Sept. 28th, 2015. We do not provide the detailed information of the CRLs domain since an adversary is able to purchase the domain from the domain re-seller and he/she can maliciously use the domain. For example, he/she is able to either revoke all code signing certificates issued by the CA or remove revoked serial numbers of compromised certificates from CLRs used to sign malicious executable files. The lesson from this case is that the root/intermediate CAs must take care of CRLs or OCSP server if leaf CAs are no longer maintained or operated.

Another example is *AOL*. AOL (https://www.aol.com/) used to issue signing certificates in the early 2000s and operated their own CRL. After terminating their certificate services, the CRLs (found under *http://crl.aol.com*) are no longer maintained and the site cannot even be reached (different than the 404 error mentioned before). Program code including malware signed with the certificates can be valid if they are trusted-timestamped since there is no way for clients to check the certificates revocation status.

We also measure the reachabilities of OCSP severs (see Section 5.2.3). In total,

15 OCSP points are unreachable by eight CAs (*AOL*, *Verisign*, *Comodo*, *StartSSL*, *WoSign*, *GlobalTrustFinder*, *Certum*, and *GlobalSign*) after removing OCSP points affected by our institution's network issues as the same as the CRLs measurements.

Forbidden, *timeout*, *bad hostname*, and *method not allowed* are the main reasons for the unreachability. In particular, *AOL* was a CA that issued code signing certificates and maintained two OCSP servers. However, AOL is no longer a CA and maintain the two OCSP servers, which leads to clients remain unable to verify AOL's code signing certificates. The unreachability of CRLs and OCSP points are very common due to various reasons as discussed above. The Windows platforms should properly handle these failures; but because the Windows platforms apply the soft-fail policy, executable files including malware signed with these code signing certificates can remain valid.

5.5.3.3 Mismanagement in CRLs and OCSPs

In this section, we highlight several mismanagement issues in the CAs side for the dissemination of revocation status information for clients.

No longer updated CRLs. According to the minimum requirements [38] for code signing CAs, CRLs should be updated and re-issued at least once a week; furthermore, the next update timestamp in the *nextUpdate* field also cannot be more than ten days from the *thisUpdate* field.

We examine how frequently CAs update and re-issue their CRLs. 57 CRLs (out of 215 CRLs) have been never updated. Most (34 of 57, 59.6%) CRLs are

issued and maintained by *Shanghai Electronic CA*; but major code signing CAs' CRLs are included in the 57 CRLs. Furthermore, most CRLs (130, 89.7% out of 145 CRLs, except for unreachable or not-updated CRLs) are updated and re-issued every day, which indicates that CAs update and re-issue their CRLs when code signing certificate is revoked and added to their CRLs.

Transient certificates in CRLs. Note that one of the distinct differences from TLS is that CAs must take care of even expired certificates due to the trusted timestamping. However, we find that 278 revoked code signing certificates are removed from 18 CRLs. The issuing CAs include *GlobalSign*, *Comodo* , *Digicert*, *Entrust*, and *Certum*. Most removed serial numbers are never re-added, but interestingly, one serial number from *Digicert* is re-added to the CRLs after 106 days.

We report this issue to all affected CAs to better understand why it happens for CAs. A CA replied back that they had a flaw in their revocation process and system that had removed expired certificates. They fixed the flaw to keep all revoked code signing certificates in the CRLs indefinitely.

Inconsistent responses from CRLs and OCSP. Most recently issued code signing certificates include CRLs and OCSP points so that clients are able to use the two mechanisms to check revocation status. Therefore, either CRLs or OCSP becomes a fallback mechanism for another one. We presumably believe that the revocation status in CRLs and OCSP should be consistent. In other words, if a revoked serial number is found in CRLs, accordingly the corresponding response from OCSP should indicate "revoked."

We observe that the revocation status responses of the 19 code signing certificates in the data set (Section 5.2.3) are inconsistent. More specifically, the code signing certificates are valid from the OCSP, but they are found in the CRLs. Of the 19 certificates, 16 and three certificate were issued by *Go Daddy* and *Starfield Technologies*). Because *Starfield* was acquired by Go Daddy, we believe that these two CAs may use the same infrastructures for revocation status information including CRLs and OCSP, which results in the inconsistency problems. It indicates that CAs must properly maintain CRLs and OCSP for consistency responses.

In Windows, OCSP is preferred over CRLs for checking revocation status. Moreover, Windows does not double-check the status using CRLs if a certificate is believed to be valid through OCSP. Therefore, inconsistent responses from OCSP and CRLs can lead to serious security problems; especially only when the responses from OCSP indicate the certificates are valid but revoked in CRLs. In this case, Windows presumably believes that binary samples signed the revoked certificates to be valid, and allows its users to execute the binary samples.

Unknown or unauthorized responses from OCSP. According to the OCSP RFC specification [28], the OCSP servers must response three statuses for a requested certificates; *good*, *revoked*, and *unknown*. *Good* means that the requested certificate is valid, *revoked* means that the certificate is revoked. Finally, *unknown* is that the requested OCSP server is unaware of the status of the requested certificate.

We examine how many OCSP servers responses the *unknown* status for requested code signing certificates. Three OCSP responders (*Certum*, *LuxTrust*, and

Shanghai Electronic CA) return that they are unaware of the 669 requested certificates issued by the corresponding CAs. Almost all of the affected code signing certificates (658, 98%) are issued by *Certum*.

OCSP servers can response with an error message due to five reasons; *malformedRequest*, *internalError*, *tryLater*, *sigRequired*, and *unauthorized*. In particular, the *unauthorized* indicates that either 1) the client is not authorized to query the OCSP server, or 2) the OCSP server cannot respond authoritatively [28]. In the OCSP responder side, they can response an *unauthorized* error either when 1) they are **unauthorized** to access the revocation status records for requested the certificates, or when 2) when the revocation status information is removed. We examine how many OCSP servers response with the *unauthorized* error messages for their issued code signing certificates. In our date set (see Section 5.2.3), we find that 2,129 code signing certificates (1.5% out of 145,582) have the *unauthorized* errors. Most of the code signing certificates (1,515, 71.2%) are issued by *Go Daddy*.

The Windows clients may not be affected by these *unauthorized* errors and *unknown* responses because the Windows platforms check CRLs when these messages are returned. However, it indicates that CAs improperly maintain their infrastructures for OCSP responders.

Chapter 6: Conclusion

Public Key Infrastructures (PKIs) enable entities to securely communicate or exchange their digital assets such as binaries. In the PKIs, two representative applications (the Web PKI and the Code-Signing PKI) are widely used. Specifically, in the Web PKI, entities (i.e., clients) are able to authenticate web servers and to securely communicate with the web servers because all messages between two entities are encrypted. In the Code-Signing PKI, software developers who want to publish their binaries in the wild sign the binaries with their private keys. In turn, clients are able to establish trust in who publish the signed binaries and the signed binaries are never altered, which means that the PKI guarantees the authenticity and integrity of binaries.

However, adversaries exploit the weaknesses in the PKIs or users' misunderstanding of the PKIs for their malicious purpose such as political or economic gains. Specifically, in the Web PKI, many modern web browsers display a green padlock icon in its URL bar when users access the websites served with valid TLS certificates, which mislead users to incorrectly understanding the meaning of the green padlock icon. They rather believe that the green padlock icon can prevent phishing attacks. Adversaries take advantage of users' misunderstanding of the green padlock icon

and launch their phishing websites with valid TLS certificates (Section 4.1). Users may be tricked by the phishing websites with valid TLS certificates due to their misunderstanding of the icons. We conduct a measurement study of the current landscape of TLS (HTTPS) phishing websites with 5.5 phishing URLs. We observe that the 1) CAs rarely revoke TLS certificates used for phishing websites and 2) a CA's vetting process fails (Section 4.1).

In the Code-Signing PKI, we first characterize the weaknesses in three main actors (CAs, publishers, and clients) that adversaries can exploit (Section 5.3). Then, we systemically measure the security threat of the Code-Signing PKI in the wild to understand the prevalence of malware signed with abusive code-signing certificates (Section 5.3). We also attempt to understand the underground economy of code-signing certificates (Section 5.4). Last, we measure the effectiveness of the primary defense, *revocation*, against the abuses (Section 5.5). We observe that the current revocation mechanism is incorrectly performed by CAs because of their misunderstanding of the Code-Signing PKI and the inherent challenge of setting effective revocation dates. These findings help a CA and Anti-Virus Engines to fix their flaws.

Bibliography

[1] Doowon Kim. Who really revokes abusive certificates? tracking the irresponsibility of certificate authorities in the wild (in submission). 2020.

[2] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraș. Certified malware: Measuring breaches of trust in the windows code-signing pki. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 2017.

[3] Kristián Kozák, Bum Jun Kwon, Doowon Kim, Christopher Gates, and Tudor Dumitraş. Issued for abuse: Measuring the underground trade in code signing certificate. In *17th Annual Workshop on the Economics of Information Security (WEIS)*, 2018.

[4] Doowon Kim, Bum Jun Kwon, Kristián Kozák, Christopher Gates, and Tudor Dumitras. The broken shield: Measuring revocation effectiveness in the windows code-signing PKI. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.

[5] 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, RFC Editor, May 2008. `http://www.rfc-editor.org/rfc/rfc5280.txt`.

[6] Google. Https encryption on the web – google transparency report. `https://transparencyreport.google.com/https/overview?hl=en`, April 2020. (Accessed on 04/23/2020).

[7] *DigiNotar SSL certificate hack amounts to cyberwar, says expert*, (accessed September 30, 2019). `https://www.theguardian.com/technology/2011/sep/05/diginotar-certificate-hack-cyberwar`.

[8] Google. Google online security blog: An update on attempted man-in-the-middle attacks. `https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html`. (Accessed on 04/20/2020).

[9] Elinor Mills. Fraudulent Google certificate points to Internet attack. *CNET*, Aug 2011.

[10] Jarno Niemela. It's Signed, therefore it's Clean, right? 2010.

[11] Mike Wood. Want My Autograph? The Use and Abuse of Digital Signatures by Malware. *Virus Bulletin Conference September 2010*, (September):1–8, 2010.

[12] Platon Kotzias, Srdjan Matic, Richard Rivera, and Juan Caballero. Certified pup: Abuse in authenticode code signing. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 465–478, New York, NY, USA, 2015. ACM.

[13] Omar Alrawi and Aziz Mohaisen. Chains of distrust: Towards understanding certificates used for signing malicious applications. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 451–456, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[14] Nicholas Falliere, Liam O'Murchu, and Eric Chien. W32.Stuxnet dossier. Symantec Whitepaper, February 2011.

[15] Zane Ma, Joshua Reynolds, Joseph Dickinson, Kaishen Wang, Taylor Judd, Joseph D Barnes, Joshua Mason, and Michael Bailey. The impact of secure transport protocols on phishing efficacy. In *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*, 2019.

[16] Katharina Krombholz, Karoline Busse, Katharina Pfeffer, Matthew Smith, and Emanuel von Zezschwitz. "If HTTPS Were Secure, I Wouldn't Need 2FA"-End User and Administrator Mental Models of HTTPS. *IEEE Security & Privacy*, 2019.

[17] Scott Ruoti, Tyler Monson, Justin Wu, Daniel Zappala, and Kent Seamons. Weighing context and trade-offs: How suburban adults selected their online security posture. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*, pages 211–228, 2017.

[18] Adrienne Porter Felt, Robert W Reeder, Alex Ainslie, Helen Harris, Max Walker, Christopher Thompson, Mustafa Embre Acer, Elisabeth Morant, and Sunny Consolvo. Rethinking connection security indicators. In *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*, pages 1–14, 2016.

[19] *Anti-Phishing Working Group: APWG Trends Report Q2 2019*, (accessed September 30, 2019). `https://docs.apwg.org/reports/apwg_trends_report_q2_2019.pdf`.

[20] Google. Https encryption on the web – google transparency report. `https://transparencyreport.google.com/https/overview?hl=en`. (Accessed on 03/08/2019).

[21] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten. Automatic certificate management environment (acme). RFC 8555, RFC Editor, March 2019.

[22] *Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates*, (accessed September 30, 2019). `https://cabforum.org/baseline-requirements-documents/`.

[23] J. Schaad, B. Kaliski, and R. Housley. Additional algorithms and identifiers for rsa cryptography for use in the internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 4055, RFC Editor, June 2005.

[24] *The Results of the CloudFlare Challenge*, (accessed September 30, 2019). `https://blog.cloudflare.com/the-results-of-the-cloudflare-challenge`.

[25] *HEARTBLEED UPDATE (V3)*, (accessed September 30, 2019). `https://blogs.akamai.com/2014/04/heartbleed-update-v3.html`.

[26] Inc. http://www. synopsys. com/ Synopsys. Heartbleed Bug, Oct 2019. [Online; accessed 30. Apr. 2020].

[27] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 debian openssl vulnerability. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 15–27, New York, NY, USA, 2009. ACM.

[28] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - ocsp. RFC 6960, RFC Editor, June 2013. `http://www.rfc-editor.org/rfc/rfc6960.txt`.

[29] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An end-to-end measurement of certificate revocation in the web's pki. In *Proceedings of the 2015 Internet Measurement Conference*, pages 183–196. ACM, 2015.

[30] Yngve N. Pettersen. The Transport Layer Security (TLS) Multiple Certificate Status Request Extension. RFC 6961, June 2013.

[31] Adam Langley, Emilia Kasper, and Ben Laurie. Certificate Transparency, Apr 2020. [Online; accessed 20. Apr. 2020].

[32] Microsoft. Windows Authenticode portable executable signature format, Mar 2008. `http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx`.

[33] Burt Kaliski. PKCS #7: Cryptographic message syntax version 1.5. RFC 2315, RFC Editor, March 1998. http://www.rfc-editor.org/rfc/rfc2315.txt.

[34] Microsoft. Windows Defender SmartScreen. https://docs.microsoft.com/en-us/windows/threat-protection/windows-defender-smartscreen/windows-defender-smartscreen-overview, 2017.

[35] Microsoft. Driver signing policy - windows drivers — microsoft docs. https://docs.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-policy--windows-vista-and-later-. (Accessed on 03/05/2019).

[36] Google. Google chrome privacy whitepaper. https://www.google.com/intl/en/chrome/privacy/whitepaper.html. (Accessed on 03/05/2019).

[37] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet x.509 public key infrastructure time-stamp protocol (tsp). RFC 3161, RFC Editor, August 2001. http://www.rfc-editor.org/rfc/rfc3161.txt.

[38] Code Signing Working Group. Minimum requirements for the issuance and management of publicly-trusted code signing certificates. Technical report, 2016.

[39] Microsoft. Code-Signing Best Practices. Technical report, 2007. https://msdn.microsoft.com/en-us/library/windows/hardware/dn653556.

[40] Rapid7. Ssl certificates — rapid7 open data. https://opendata.rapid7.com/sonar.ssl/. (Accessed on 03/10/2019).

[41] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 605–620, Berkeley, CA, USA, 2013. USENIX Association.

[42] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex 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.

[43] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitras, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 489–502, New York, NY, USA, 2014. ACM.

[44] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. The matter of Heartbleed. In *Proceedings of the 2014*

Conference on Internet Measurement Conference, IMC '14, pages 475–488, New York, NY, USA, 2014. ACM.

[45] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. Measurement and analysis of private key sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 628–640, New York, NY, USA, 2016. ACM.

[46] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. pages 183–196. ACM Press, 2015.

[47] Ponnurangam Kumaraguru, Yong Rhee, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Protecting people from phishing: The design and evaluation of an embedded training email system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, page 905–914, New York, NY, USA, 2007. Association for Computing Machinery.

[48] B. Reaves, N. Scaife, D. Tian, L. Blue, P. Traynor, and K. R. B. Butler. Sending out an sms: Characterizing the security of the sms ecosystem with public gateways. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 339–356, 2016.

[49] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. Towards detecting compromised accounts on social networks. *IEEE Transactions on Dependable and Secure Computing*, 14(4):447–460, 2017.

[50] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a haystack: Tracking down elite phishing domains in the wild. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, pages 429–442, New York, NY, USA, 2018. ACM.

[51] Nick Nikiforakis, Steven Van Acker, Wannes Meert, Lieven Desmet, Frank Piessens, and Wouter Joosen. Bitsquatting: Exploiting bit-flips for fun, or profit? In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 989–998, New York, NY, USA, 2013. ACM.

[52] Tobias Holgers, David E. Watson, and Steven D. Gribble. Cutting through the confusion: A measurement study of homograph attacks. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 24–24, Berkeley, CA, USA, 2006. USENIX Association.

[53] Panagiotis Kintis, Najmeh Miramirkhani, Charles Lever, Yizheng Chen, Rosa Romero-Gómez, Nikolaos Pitropakis, Nick Nikiforakis, and Manos Antonakakis. Hiding in plain sight: A longitudinal study of combosquatting abuse.

In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 569–586, New York, NY, USA, 2017. ACM.

[54] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a haystack: Tracking down elite phishing domains in the wild. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 429–442, New York, NY, USA, 2018. Association for Computing Machinery.

[55] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1143–1155, New York, NY, USA, 2014. ACM.

[56] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2, 2008.

[57] Eric Rescorla. The transport layer security (tls) protocol version 1.3, 2018.

[58] *Anti-Phishing Working Group*, (accessed September 30, 2019). `https://apwg.org`.

[59] Peng Peng, Chao Xu, Luke Quinn, Hang Hu, Bimal Viswanath, and Gang Wang. What happens after you leak your password: Understanding credential sharing on phishing sites. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, pages 181–192, New York, NY, USA, 2019. ACM.

[60] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, RFC Editor, June 2013.

[61] Adam Oest, Yeganeh Safaei, Adam Doupe, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. PhishFarm: A Scalable Framework for Measuring the Effectiveness of Evasion Techniques against Browser Phishing Blacklists. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1344–1361. IEEE, 2019.

[62] Doowon Kim, Bum Jun Kwon, Kristián Kozák, Christopher Gates, and Tudor Dumitras. The broken shield: Measuring revocation effectiveness in the windows code-signing PKI. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 851–868, Baltimore, MD, August 2018. USENIX Association.

[63] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. Tracking certificate misissuance in the wild. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 785–798. IEEE, 2018.

[64] DAN GOODIN. Stuxnet spawn infected kaspersky using stolen foxconn digital certificates, Jun 2015.

[65] Swiat. Flame malware collision attack explained, Jun 2012.

[66] Microsoft. Erroneous verisign-issued digital certificates pose spoofing hazard, 2001.

[67] Tudor Dumitraș and Darren Shou. Toward a standard benchmark for computer security research: The Worldwide Intelligence Network Environment (WINE). In *EuroSys BADGERS Workshop*, Salzburg, Austria, Apr 2011.

[68] VirusTotal. `www.virustotal.com`, 2017.

[69] Bum Jun Kwon, Virinchi Srinivas, Amol Deshpande, and Tudor Dumitraș. Catching worms, trojan horses and pups: Unsupervised detection of silent delivery campaigns. In *Proc. NDSS*, 2017.

[70] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring pup prevalence and pup distribution through pay-per-install services. In *Proceedings of the USENIX Security Symposium*, 2016.

[71] Kurt Thomas, Juan A. Elices Crespo, Ryan Rasti, Jean Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, Lucas Ballard, Robert Shield, Nav Jagpal, Moheeb Abu Rajab, Panayiotis Mavrommatis, Niels Provos, Elie Bursztein, and Damon McCoy. Investigating commercial pay-per-install and the distribution of unwanted software. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 721–739, 2016.

[72] Evangelos E. Papalexakis, Tudor Dumitras, Duen Horng (Polo) Chau, B. Aditya Prakash, and Christos Faloutsos. Spatio-temporal mining of software adoption & penetration. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, Niagara Falls, CA, Aug 2103.

[73] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by internet-wide scanning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 542–553, New York, NY, USA, 2015. ACM.

[74] Eric Lawrence. Everything you need to know about authenticode code signing, Mar 2011.

[75] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer, 2016.

[76] David. G. Kleinbaum and Mitchell Klein. *Survival Analysis: A Self-Learning Text*. Springer, 3 edition, 2011.

[77] InfoArmor. GovRAT – Advanced Persistent Threads: Digital certificates on sale in the underground. `http://www.infoarmor.com/wp-content/uploads/2016/04/Advanced_Persistent_Threats_Code_Signing.pdf`, 2016.

[78] Limor Kessem. Certificates-as-a-service? code signing certs become popular cybercrime commodity. `https://securityintelligence.com/certificates-as-a-service-code-signing-certs-become-popular-cybercrime-commod` 2015.

[79] Charlie Osborne. Software code signing certificates worth more than guns on the dark web. `http://www.zdnet.com/article/illicit-certificates-worth-more-than-guns-on-the-dark-web/`, Oct 2017.

[80] Nicolas Christin. Traveling the silk road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web*, pages 213–224. ACM, 2013.

[81] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL landscape: a thorough analysis of the X.509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 427–444. ACM, 2011.

[82] Charles J Krebs et al. Ecological methodology. Technical report, Harper & Row New York, 1989.