Certified PUP: Abuse in Authenticode Code Signing

Platon Kotzias
IMDEA Software Institute &
Universidad Politécnica de
Madrid, Spain
platon.kotzias@imdea.org

Srdjan Matic
Universita degli Studi di Milano
Milan, Italy
srdjan.matic@unimi.it

Richard Rivera
IMDEA Software Institute &
Universidad Politécnica de
Madrid, Spain
richard.rivera@imdea.org

Juan Caballero
IMDEA Software Institute
Madrid, Spain
juan.caballero@imdea.org

ABSTRACT

Code signing is a solution to verify the integrity of software and its publisher's identity, but it can be abused by malware and potentially unwanted programs (PUP) to look benign. This work performs a systematic analysis of Windows Authenticode code signing abuse, evaluating the effectiveness of existing defenses by certification authorities. We identify a problematic scenario in Authenticode where timestamped signed malware successfully validates even after the revocation of their code signing certificate. We propose hard revocations as a solution. We build an infrastructure that automatically analyzes potentially malicious executables, selects those signed, clusters them into operations, determines if they are PUP or malware, and produces a certificate blacklist.

We use our infrastructure to evaluate 356 K samples from 2006-2015. Our analysis shows that most signed samples are PUP (88%–95%) and that malware is not commonly signed (5%–12%). We observe PUP rapidly increasing over time in our corpus. We measure the effectiveness of CA defenses such as identity checks and revocation, finding that 99.8% of signed PUP and 37% of signed malware use CA-issued certificates and only 17% of malware certificates and 15% of PUP certificates have been revoked. We observe most revocations lack an accurate revocation reason. We analyze the code signing infrastructure of the 10 largest PUP operations exposing that they heavily use file and certificate polymorphism and that 7 of them have multiple certificates revoked. Our infrastructure also generates a certificate blacklist 9x larger than current ones.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

Windows Authenticode; Code Signing; PUP; Malware

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813665.

1. INTRODUCTION

Publishers of malicious software (malware) and potentially unwanted programs (PUP) are always looking for ways to make their code look benign in order to convince the user to install it and avoid detection. One such way is *code signing*, where the software is distributed with a digital signature which, if valid, certifies the integrity of the software and the identity of the publisher. Signed code looks more benign and may be assigned higher reputation by security products. In Windows, properly signed application code avoids scary warnings when a user executes it and is assigned higher reputation when downloaded through Internet Explorer [34]. Furthermore, kernel-mode code is required to be signed. Aware of these benefits attackers are increasingly leveraging signed code for their goals, e.g., for launching notorious targeted attacks [8, 12, 13].

To sign Windows programs, publishers need to obtain a valid *code signing certificate* from a Certification Authority (CA). This should pose a barrier for malicious software, since it requires providing the publisher's identity to the CA and paying a fee ($60–$500 for 1-year certificates). Furthermore, when malicious software is observed in the wild signed with a valid certificate, the CA that issued the certificate should swiftly revoke it. However, it is not clear how well defenses such as identity checks and revocation work. Prior work in 2010 by two AV vendors [37, 41] showed that signed samples were not uncommon in malware datasets. But, there has been no systematic study analyzing the extent to which malware (e.g., bots, trojans) and PUP (e.g., adware, bundles) are abusing code signing and how well defenses such as identity validation and revocation work.

In this work we perform a systematic study on abuse of Windows *Authenticode* [31] code signing. We identify a problematic interaction between revocation and timestamping in Authenticode, where timestamped signed executables still validate even if their code signing certificate is revoked. To address this issue we propose that CAs perform *hard revocations* that invalidate all executables signed by a certificate.

We build an infrastructure that takes as input a large number of potentially malicious samples, filters out benign samples and those that are not signed, and thoroughly analyzes signed samples including their digital signatures, certificate chains, certificate revocation, and file timestamping (i.e., third-party certification of the time they saw some signed code). It also clusters signed samples into operations and classifies them as PUP or malware. Our infrastructure automatically builds a blacklist of malicious certificates, which can be used by CAs to perform revocation, or users can embed it into the Windows untrusted certificate store to block malicious code.

Using our infrastructure we analyze 356 K malware samples distributed between 2006 and February 2015, of which 142 K (42%) are signed. This process outputs a blacklist of over 2,170 code signing certificates, 9x larger than existing blacklists [5].

Our analysis uncovers that most signed samples are PUP (88%–95%) and that malware is not commonly signed (5%–12%). We observe PUP rapidly increasing over time in our corpus, reaching 88% of the samples in 2014. We measure the effectiveness of CA defenses such as identity checks and revocation. We find that 99.8% of signed PUP and 37% of signed malware use CA-issued certificates indicating that CA identity checks pose some barrier to malware, but do not affect PUP. Only 17% of malware certificates and 15% of PUP certificates have been revoked, and the best CA revokes 43% of the certificates it issues to malware and PUP publishers. Most CAs do not provide abuse email addresses and do not accurately report the revocation reason. Only 53% of the revocations include a revocation reason and those with one often report key compromise even if it is a malware-abused certificate.

Our clustering of signed samples into operations and the classification into PUP and malware shows that the largest operations correspond to PUP, e.g., adware and gray pay-per-install programs that offer users to install third-party programs. We analyze the 10 largest PUP operations observing that they heavily use polymorphism in files and certificates, possibly to bypass AV and CA checks. Seven of them have multiple certificates revoked, so CAs seem to consider them malicious. To achieve certificate polymorphism, PUP publishers buy certificates from multiple CAs, modify the Subject information, and use multiple companies and individuals. For example, OutBrowse uses 40 different companies across 6 countries to obtain 97 code signing certificates from 5 CAs.

We also leverage the fact that timestamped malware contains a trusted timestamp close to its creation to evaluate how fast VirusTotal [15], a large malware repository, collects malware.

Contributions:

- We perform a systematic analysis of Authenticode abuse and the effectiveness of existing defenses. We identify a problematic scenario in Authenticode where timestamped signed malware successfully validates even after their code signing certificate has been revoked. To address this issue we propose that CAs perform a hard revocation, which invalidates any code signed with a certificate after this has been revoked.

- We propose a novel clustering of signed samples into operations using static features extracted from the Authenticode data. We also propose two novel techniques to classify samples as PUP or malware based on the AV detection labels.

- We build an infrastructure that given large amounts of potentially malicious software automatically analyzes signed samples, clusters them into operations, classifies them as PUP or malware, and produces a blacklist of malicious certificates.

- We use our infrastructure to analyze 356 K samples. We observe that PUP is rapidly increasing, most signed samples are PUP, and malware is not commonly signed. We measure that 99.8% of signed PUP and 37% of signed malware use CA-issued certificates and only 17% of malware certificates and 15% of PUP certificates have been revoked. Most revocations lack an accurate revocation reason. We analyze the largest PUP operations exposing that they heavily use file and certificate polymorphism. In addition, most of the largest operations have multiple certificates revoked that indicates that CAs consider them malicious.

- We leverage timestamped malware to evaluate the speed with which the VirusTotal online service collects malware.

- We setup a website for our blacklist and analysis results [7].

2. OVERVIEW

Code signing is the process of digitally signing executable code and scripts. It authenticates the code's publisher and guarantees the integrity of the code. Code signing is used with different types of code in a variety of platforms including Windows executables and kernel drivers, Java JAR files, Android applications, active code in Microsoft Office documents, Firefox extensions, Adobe Air applications, and iOS applications.

The code signing process first computes a hash of the code and then digitally signs this hash using the publisher's private key. The public key of the code's publisher is authenticated using a X509 *code signing certificate* that a certification authority (CA) issues to the publisher after verifying its identity. This code signing certificate is attached to the signed code. The CA also provides its certificate chain, anchored at a trusted root CA. This chain is attached to the signed code or made available online.

In code signing, certificates are distributed with the signed code (e.g., embedded in the executable file) to geographically distributed users. When a certificate expires, it is difficult to update all code installations with a new certificate. In contrast, Web servers can simply update their HTTPS certificate between sessions. To address this issue, some code signing solutions (e.g., Windows Authenticode, Java) introduce an optional timestamping process, that sends the signed code to a Time Stamping Authority (TSA), which certifies that it observed the signed code at a specific time.

Usually, when the code signing certificate expires, validation fails. But, if the signed code is also properly timestamped within the validity period of the code signing certificate, validation succeeds despite the code signing certificate having expired.

To timestamp signed code, the TSA embeds a timestamp, digitally (counter)signs both the timestamp and the existing code signature using its private key, and authenticates its public key by including its certificate chain anchored at a trusted root CA. Thus, code that is timestamped contains two certificate chains: the *signing chain* and the *timestamping chain*.

Figure 1 summarizes the code signing process. A (potentially malicious) publisher buys a code signing certificate from a CA that verifies the publisher's identity before issuing the certificate (❶,❷). The publisher signs its code using the code signing certificate and a signing tool like Microsoft's AuthTool (❸,❹). Optionally, the publisher sends the signed code to the TSA to be timestamped (❺,❻). Finally, the publisher distributes the code to the users (❼).

2.1 Microsoft Authenticode

Authenticode is a code signing solution by Microsoft [31]. It was introduced with Windows 2000, but its specification was not publicly released until March 2008. It uses a Public-Key Cryptography Standards (PKCS) #7 SignedData structure [27] and X.509 v3 certificates [21] to bind an Authenticode-signed file to a publisher's identity. Authenticode is used to digitally sign portable executable (PE) files including executables (.exe), dynamically loaded libraries (.dll), and device drivers (.sys). It can also be used for signing Active X controls (.ocx), installation (.msi), or cabinet (.cab) files.

File format. Figure 2 presents the basic format of an Authenticode-signed PE file. It contains a PKCS #7 SignedData structure (also called Authenticode signature) at the end of the file, whose start-

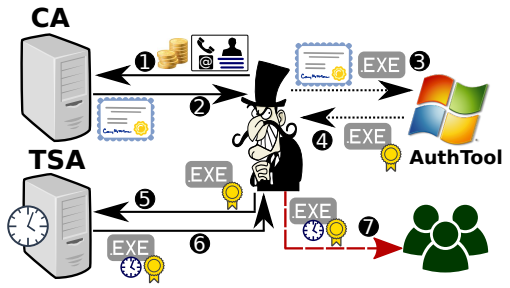

Figure 1: Code signing process: ❶,❷ publisher acquires a code signing certificate providing its personal information; ❸,❹ publisher signs code; ❺,❻ (optional) publisher submits the signed code to be timestamped; ❼ publisher distributes the signed (and timestamped) code.

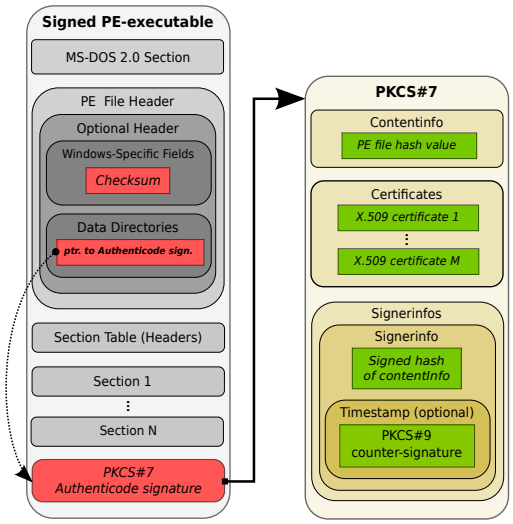

Figure 2: Format of a signed PE file. The red text box fields are not included in the calculation of the digest.

ing offset and size are captured in the `Certificate Table` field in the `Optional Header`. The PKCS #7 structure contains the PE file's hash, the digital signature of the hash generated with the publisher's private key, and the certificates comprising the signing chain (the root certificate does not need to be included). It can also optionally include a description of the software publisher, a URL, and a timestamp. Authenticode only supports MD5 and SHA1 hashes and prior work has shown how to produce Authenticode collisions with MD5 [39].

When calculating the hash of the PE file 3 fields are skipped (marked in red in Figure 2): the Authenticode signature itself, the file's checksum, and the pointer to the Authenticode signature. In addition, the PE sections are sorted before adding them to the hash. We call the result *Authentihash* to distinguish it from the file hash that includes all bytes in a file and is often used to uniquely identify a file (e.g., by security vendors). In the past, vulnerabilities have been disclosed where attackers could embed data in unspecified PE fields [25] and the Authenticode signature [30] without invalidating the file's signature.

Timestamping. Timestamping is optional in Authenticode. In order to timestamp an Authenticode-signed file, a TSA first needs to obtained the current UTC timestamp. Then, it builds a PKCS #9 counter-signature by digitally signing with its private key the timestamp and the hash of the file's signature in the PKCS #7. Next, it embeds into the PKCS #7 `SignerInfo` structure the timestamp and the counter-signature. If the optional timestamp field already existed, it is overwritten. Finally, it appends the certificates of the timestamping chain to the `certificates` part (the root certificate of the timestamping chain does not need to be included).

Revocation. Certificates can be revoked, e.g., if the private key corresponding to the public key in the certificate is compromised, using certificate revocation lists (CRLs) [21] and the online certificate status protocol (OCSP) [38].

Validation. Authenticode validation is performed using the WinVerifyTrust function, which supports multiple validation policies. The policy we are interested in is the default one for Windows (WINTRUST_ACTION_GENERIC_VERIFY_V2). This policy is documented in the Authenticode specification [31] as follows:

- The signing chain must be built to a trusted root certificate (in the Windows Certificate Store) following RFC 5280 [21].
- The signing certificate must contain either the extended key usage (EKU) CODE_SIGNING value, or the entire certificate chain must contain no EKUs.
- The certificates in the signing chain must not be in the untrusted certificates store[1].
- Each certificate in the signing chain must be within its validity period, or the signature must be timestamped.
- Revocation checking is optional, but often used.
- The timestamping chain validation differs in that the TSA certificate must include a TIMESTAMP_SIGNING EKU and revocation is turned off by default for this chain.
- By default, timestamping extends the lifetime of the signature indefinitely, as long as it happened during the validity period of the signing certificate and before the certificate revocation date (if applicable).
- Timestamped signatures can be prevented from verifying for an indefinite period of time by setting the LIFETIME_SIGNING OID in the code signing certificate or passing a particular flag to the WinVerifyTrust function.
- The Authenticode signature must verify.
- The Authentihash computed on the executable must equal the Authentihash value stored in the PKCS #7 structure.

A failure in any of these steps should cause validation to fail. Unfortunately, the Authenticode validation code is proprietary and thus it is not clear if it follows all steps in the validation, in which order those steps are executed, and how it handles cases where the specification is unclear. Its exact functionality can only be reverse-engineering through testing or code analysis. We discuss validation issues in Section 3.

Code signing in Windows. A signed executable can embed an Authenticode signature (Figure 2) or its hash can be included in a *catalog file*, i.e., a collection of file hashes digitally signed by their publisher [4]. Most non-Microsoft signed executables embed Authenticode signatures. By default, user-level signed applications are validated by Windows before they run if the application was

[1]In Windows XP and 2003 only the signing certificate is checked.

		Certificate Price		Revocation		
CA	**TSA**	**CS**	**HTTPS**	**Mal.**	**Abuse**	**Delay**
Certum	✓	$199	$34	-	-	$\leq$ 1d
Comodo	✓	$172	$109	✓	✓	$\leq$ 1d
DigiCert	✓	$223	$175	✓	✓	$\leq$ 1d
Disig	-	$109	$51	✓	✓	$\leq$ 1d
Entrust	✓	$299	$199	✓	-	$\leq$ 1d
GlobalSign	✓	$299	$249	-	-	$\leq$ 3h
GoDaddy/StarField	✓	$170	$63	-	-	$\leq$ 7d
StartCom/StartSSL	✓	$60	$60	-	-	$\leq$ 12h
SwissSign	✓	$449	$399	-	-	$\leq$ 1d
Symantec/GeoTrust	✓	$499	$149	-	-	$\leq$ 1d
Symantec/Thawte	-	$299	$149	-	-	$\leq$ 1d
Symantec/Verisign	✓	$499	$399	-	-	$\leq$ 1d
TrustWave	-	$329	$119	✓	-	$\leq$ 1d
TurkTrust	-	$138	$112	✓	-	$\leq$ 1d
Verizon	-	$349	$349	-	-	$\leq$ 12h
WoSign	✓	$466	$949	-	✓	$\leq$ 1d
yessign	-	$153	-	-	-	$\leq$ 1d

Table 1: CAs offering code signing certificates and timestamping. Prices are for 1-year certificates in US Dollars. Revocation shows if a malware clause is present in the CPS, an abuse contact is mentioned, and the delay to publish a revocation. A dash indicates that we were not able to find related information.

downloaded from the network (including network shares) or requires administrator privileges, which triggers User Account Control (UAC). In addition, Internet Explorer validates the signature of downloaded files [10]. User interaction varies across situations and Windows versions, but generally if the Authenticode signature validates, the window presented to the user to confirm execution contains the verified publisher information and a warning icon. If it fails or is unsigned it states the publisher is untrusted and uses a more threatening icon and textual description. Since Windows 7, AppLocker allows specifying rules for which users or groups can run particular applications, which allows to create a rule for running only signed applications [1].

Device drivers are handled differently depending on the Windows version, whether 32-bit or 64-bit, and if the driver is user-mode or kernel-mode [33]. For 64-bit Windows since Vista, it's mandatory to have both user-mode and kernel-mode drivers signed in order to load. In addition, for kernel-mode code a special process is required where the publisher's code signing certificate must have a chain leading to the Microsoft Code Verification Root [6].

2.2 Authenticode Market

We analyzed the CAs that are members of the CA Security Council [2] and the CA/Browser [3] forum and that publicly sell Authenticode code signing certificates. Table 1 summarizes if they offer timestamping services, their certificate prices, and their revocation policies.

Few CAs offer Authenticode code signing certificates compared to HTTPS certificates, possibly reflecting a smaller market. There has been significant consolidation, e.g., Symantec acquired Verisign, GeoTrust, Thwate, and smaller CAs. Unfortunately, we did not find any public market size and CA market share figures. Only 11 CAs publicly advertise themselves as TSAs. In all cases timestamping is offered through HTTP, free of charge, and does not require authenticating to the service. We evaluate these services in Section 5.6.

Code signing certificates are pricier than HTTPS certificates ranging $60–$499 for a 1-year validity period. The exception is StartSSL which charges per identity verification rather than per certificate. Code signing certificates can be bought with a 1, 2, or 3-year validity period, with the latter being offered by only 25% of the CAs.

Revocation. We examine the revocation sections of the Certification Practice Statement (CPS) documents of the CAs in Table 1. The only entity that can perform revocation is the same CA that issued the code signing certificate. For all CAs, the customer can request revocation of its own certificate and the delay to publish the revocation through CRL or OCSP ranges between 3 hours and a week (Delay column). Only 6 CAs have a specific clause on their CPS about revocation being possible if the code signing certificate is abused to sign malicious code (Mal. column), although there is typically another clause that reminds how revocation can happen if the certificate is used in a way "harmful for the image or the business" of the CA. We were able to find an abuse contact email address or Web form for only 4 CAs. In other cases third parties reporting abuse would have to go through generic contact forms. Researchers that in the past requested CAs to revoke malicious certificates reported none or little response [37, 41]. Overall, third-party reporting of certificate abuse does not seem a concern by most CAs. In Section 5.5 we show that only 15% of the malicious certificates we observe are revoked, and the revocation practices outlined in this paragraph are likely a contributing factor for this low number.

3. REVOKING TIMESTAMPED CODE

When a code signing certificate is revoked, any executable signed (but not timestamped) with this certificate no longer validates in Windows, regardless if the executable was signed before or after revocation. But, our testing of Windows Authenticode validation reveals that if an executable is both properly signed and timestamped at time t_{ts} (i.e., validates at t_{ts}) and then its code signing certificate is revoked at $t_{rev} > t_{ts}$, the executable still validates at any $t > t_{rev}$ despite the code signing certificate having been revoked. This is true as long as the revocation date is larger than the timestamping date ($t_{rev} > t_{ts}$). Executables timestamped after the revocation ($t_{rev} \leq t_{ts}$) will not validate.

Such handling seems to assume that revocation happens because the private key corresponding to the certificate's public key was compromised. In that case, executables signed before the key compromise should be OK and there is no need for them to fail validation, as long as a timestamp certifies they existed before revocation. We call this a *soft revocation* because it only invalidates executables signed with the certificate after revocation.

However, revocation is also needed when an attacker convinces a CA to issue him a code signing certificate, which it uses only to sign malicious code. In this case, if the attacker signs and timestamps a large number of malware before starting to distribute them, revocation happens after the timestamping date of those samples and thus they still validate after the revocation. What is needed in this case is a *hard revocation* that invalidates all executables signed with that certificate regardless when they were timestamped. With a hard revocation the CA sends the signal that it believes the certificate's owner is using it for malicious purposes and none of his executables should be trusted, rather than the owner was compromised and earlier signed executables are OK. The CA is responsible for distinguishing these two cases.

The easiest way to perform a hard revocation is for the CA to set the revocation date to the certificate's issue date ($t_{rev} = t_i$), even if the CA discovers the improper use of the certificate at a later time. This way, any sample signed with that certificate will fail validation, regardless the timestamping date, because the revocation date will always be smaller than the timestamping date. Note that the

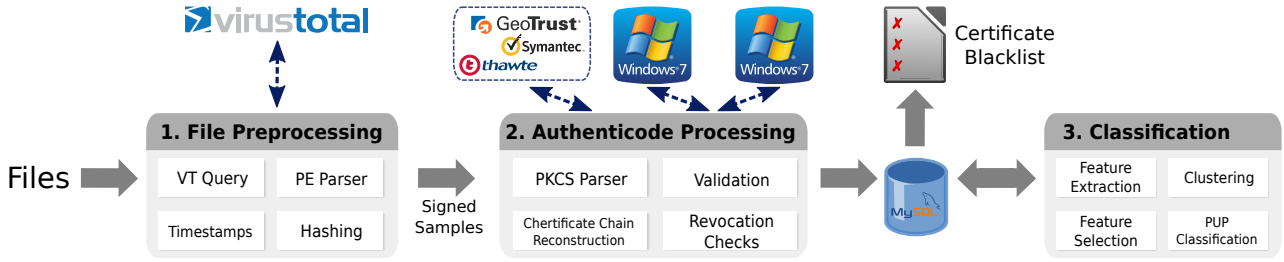

Figure 3: Approach overview.

attacker cannot forge an old timestamp and also that a valid timestamp needs to be within the certificate's validity period.

Setting the revocation date to the certificate's issue date enables hard revocations without modifying OCSP and CRLs. A caveat is that it hides the real date in which the CA realized the certificate was malicious. Adding this information may require modifications to revocation protocols and CAs may see a benefit on hiding how long it takes them to realize they issued a malicious certificate. We believe that it would be good to use the OCSP/CRL revocation reason to explicitly state that it is a hard revocation. In Section 5 we show that the information in this field is currently not useful.

One issue is that an attacker could revoke its own certificate after claiming a key compromise, in order for the CA to perform a soft revocation that does not invalidate previous code. One way to address this is to assign reputation to subjects based on prior revocations. In Section 6 we discuss that CAs should share revocation information.

We have reported to Microsoft this issue and the suggested solution using hard revocations.

4. APPROACH

Figure 3 summarizes our approach. It takes as input a large number of unlabeled files from malware datasets (described in Section 5.1). It preprocesses the files to discard benign files, parses the PE files to identify those signed, and processes the Authenticode signature (Section 4.1). All information is stored in a central database. Then, the clustering (Section 4.2) groups the samples into operations. Finally, the certificate blacklist is output. For each blacklisted certificate we provide information about the certificate (i.e., Subject CN, Issuer, Serial Number), a link to VirusTotal with a sample signed with this certificate, and the certificate itself exported in DER format that can be directly installed on Windows untrusted certificates store by following the Windows certificate installation wizard.

4.1 Sample Processing

Our infrastructure is implemented on Linux using 4,411 lines of Python and C code. Files are first preprocessed to remove non-PE files. Then, the PE files are parsed to extract a variety of information from the PE header including the file type (EXE, DLL, SYS), multiples hashes (MD5, SHA1, SHA256, PEHash [40]), publisher and product information in the optional PE structures, icon, PDB path, and a number of timestamps. Next, it queries the file hash to VirusTotal (VT) [15], an online service that examines malware with a large number of security tools, to retrieve file metadata such as the number of AV engines that detect the file and the timestamp of the first time the file was submitted. We keep any sample flagged by more than 3 AV engines. Samples that contain an Authenticode signature move on to the next processing phase.

The Authenticode processing parses the PKCS #7 structure to retrieve the code signature, timestamp, and PKCS #9 timestamping counter-signature (if present). Then, it extracts the X.509 certificates from the `certificates` structure of the PKCS #7 structure, which contains certificates from both the signing and timestamping chains. The certificate chains need to be reconstructed because oftentimes the certificates are not properly ordered and certificates from both chains may be mixed. In addition, certificates can include a URL to the next certificate in the chains (if not included). If so, the certificate is downloaded. Next, the certificates are parsed to obtain a wealth of information including among others, the Subject, Issuer, validity period, PEM and DER hashes, Extended Key Usage flags, and OCSP and CRL URLs. The validation component verifies both chains using OpenSSL and queues the files to be distributed across four Windows VMs for Authenticode validation. We use the OpenSSL validation to better understand the error codes returned by Authenticode validation. Next, the revocation component retrieves and processes the CRL and OCSP information from each certificate in the chain. All information is stored in a central database.

4.2 Clustering

We cluster the signed samples into operations by grouping executables from the same publisher. For computing the sample similarity we focus on features that can be extracted statically from the samples, which enables efficient processing. Since all 142 K samples to be clustered are signed, most of our features focus on properties of the publisher's code signing certificate, with a focus on identifying different certificates from the same publisher. As far as we know certificate features have not been previously used for clustering malware. We also use a previously proposed static feature to identify polymorphic variants of the same code [40].

We first identify a large set of candidate features and perform feature selection on the signed samples of the publicly available Malicia malware dataset [35], for which the majority of files have family labels. We select the following 6 top boolean features based on information gain:

- **Leaf certificate.** Properly signed samples using the same CA-issued code signing certificate (same certificate hash) are distributed by the same publisher, i.e., the one owning the certificate. Publishers typically amortize the cost of a certificate by signing a large number of samples.

- **Leaf certificate public key.** Public keys are left unchanged in many certificate replacements [36]. Thus, two certificates authenticating the same public key likely belong to the same publisher.

- **Authentihash.** Files with the same Authentihash contain the same code and data. Thus, they correspond to the same pro-

Samples	Families	Precision	Recall	F-Measure
2,046	7	98.6%	33.2%	49.7%

Table 2: Clustering accuracy on labeled (signed) malware from Malicia dataset.

gram even if they have a different file hash, e.g., due to different certificate chains.

- **Subject common name.** Publishers may try to obtain multiple certificates using the same identity by slightly modifying the company or individual name (e.g., "Company SLU" and "Company S.L."). Given two certificates with a non-empty Subject CN field, this feature computes the normalized edit distance between their Subject CNs. If the distance is less than 0.11 their publishers are considered the same. The threshold is chosen using a small subset of manually labeled certificates.

- **Subject location.** Publishers may reuse the same address in multiple certificates with small changes to fool the CA (e.g., "Rockscheld Blvd. 83 Dublin" and "Rockchilde 83 Dublin"). Given two certificates whose subject location contains a street attribute, this feature computes the normalized edit distance between those fields. If less than 0.27 the publisher is the same, otherwise different. The threshold is chosen using a small subset of manually labeled certificates. If the street attribute is not available, then the location only has the city and is not specific enough, thus they are considered different.

- **File metadata.** PE executables have an optional data structure with file metadata. This feature concatenates the following file metadata fields: publisher, description, internal name, original name, product name, copyright, and trademarks. Two files with the same concatenated metadata string larger than 14 characters are considered to be in the same family. Shorter metadata strings are not specific enough, thus they are considered different.

- **PEhash.** We also use the previously proposed PEhash [40], which transforms structural information about a PE executable into a hash value. If two files have an unknown packer and the same PEhash they are considered polymorphic variants of the same code.

Clustering. We use the following algorithm to cluster files into operations. The clustering starts with zero clusters and iterates on the list of samples. For each sample, it checks if it is similar to any other sample using the 6 features above. Two samples are similar if any of the above similarity features returns one. If the sample being examined is similar only to samples in the same cluster, it is added to that cluster. If similar to samples in multiple clusters, those clusters are merged and the sample is added to the merged cluster. If not similar to any other sample, a new singleton cluster is created for it.

Clustering accuracy. To evaluate the accuracy of our clustering we use the publicly available Malicia malware dataset [35], which contains labeled samples. In particular we use the 2,046 samples in the Malicia dataset that are both signed and have a label. Those samples belong to 7 families, but the majority (97%) are Zbot. Table 2 summarizes the results. The precision is high (98.6%) but the recall is low (33.2%). The reason for the low recall is that the Malicia labels capture samples with the same code. However, Zbot code can be bought or downloaded online, so it is used by many operations. Our clustering is oriented towards different operations so Zbot is broken into multiple clusters.

Labeling. Our clustering automatically generates a cluster label based on the most common feature value in the cluster. For the largest clusters we manually update the label with any popular tag used by security vendors for that operation.

4.3 PUP classification

We are interested in differentiating how malware and PUP abuse Authenticode, but are not aware of any prior techniques to automatically differentiate both classes. The main challenge is that the behaviors used to determine if a family is potentially unwanted or malicious may differ across security vendors [9, 16]. To address this issue we design two techniques that examine the AV detection labels obtained from VirusTotal, taking into account how multiple AV engines classify samples as PUP or not. One technique classifies a whole cluster as PUP or malware, while the other classifies each sample separately. We find the cluster classification to be more accurate, but it requires the clustering in Section 4.2, which is only available for signed samples and cannot be applied to unsigned samples as most features come from the certificates. We use the sample classification to compare the PUP prevalence among signed and unsigned samples.

Prior work has shown that AV labels are not a good ground truth for classifying malware into families due to inconsistent naming [18, 32]. However, our classification is at a coarser granularity. We only use the AV labels to determine if a cluster or a sample corresponds to PUP or not rather than to a specific family, which is captured by the malware clustering in Section 4.2.

As preparation for both classification techniques we first select 13 case-insensitive keywords that if present in a label indicate a potentially unwanted program: PUP, PUA, adware, grayware, riskware, not-a-virus, unwanted, unwnt, toolbar, adload, adknowledge, casino, and casonline. Then, we select the top 11 AV engines sorted by number of samples in all our datasets whose detection label includes at least one of the 13 above keywords. Those engines are: Malwarebytes, K7AntiVirus, Avast, AhnLab-V3, Kaspersky, K7GW, Ikarus, Fortinet, Antiy-AVL, Agnitum, and ESET-NOD32.

Using the selected keywords and AV engines, the classification module automatically classifies each cluster or sample as PUP or malware. Both classifications perform a majority voting on whether the selected engines consider the cluster or sample as PUP or not. We detail them next.

Cluster classification. The cluster classification first obtains for every engine the most common label the engine outputs on samples in the cluster (engines often output multiple labels for samples in the same cluster). Then, if the most common label for an engine contains at least one of the 13 keywords, the PUP counter is increased by one, otherwise the malware counter is increased by one. After evaluating all 11 engines on the cluster, if the PUP counter is larger or equal to the malware counter the cluster is considered PUP, otherwise malware.

Sample classification. The sample classification gets the label of the selected 11 engines for the sample. It can happen that some (and even all) of the selected engines do not detect the sample. If the label for an engine contains at least one of the 13 keywords, the PUP counter is increased by one, otherwise the malware counter is increased by one. After evaluating the labels, if the PUP counter is larger or equal to the malware counter the cluster is considered PUP, otherwise malware.

		PE		CS chain				TS chain			
Dataset	**Date**	**All**	**Malware+PUP**	**PE Signed**	**PUP**	**Chains**	**Leaf**	**PE Timestamped**	**PUP**	**Chains**	**Leaf**
CCSS	05/2015	197	191 (97.0%)	172 (90.0%)	6.4%	172	171	92 (53.5%)	6.5%	18	16
VirusShare_149	02/2015	32,184	30,402 (94.5%)	19,082 (62.8%)	97.4%	855	815	7,419 (38.8%)	96.0%	32	24
VirusShare_148	02/2015	64,629	59,684 (92.3%)	45,668 (76.5%)	97.6%	1,077	1,015	15,059 (32.9%)	96.6%	34	24
VirusShare_138	08/2014	53,064	51,500 (97.0%)	46,174 (89.7%)	99.2%	684	656	29,491 (63.9%)	99.1%	28	21
NetCrypt	08/2014	1,052	1,051 (99.9%)	892 (84.9%)	99.5%	28	26	8 (0.9%)	62.5%	3	3
VirusShare_99	09/2013	99,616	96,355 (96.7%)	26,424 (27.4%)	92.5%	1,057	990	7,002 (26.5%)	90.8%	46	33
Malicia	05/2013	11,337	11,333 (99.9%)	2,059 (18.2%)	0%	87	87	2 (0.10%)	0%	1	1
VirusShare_0	06/2012	87,126	86,112 (98.8%)	1,906 (2.2%)	56.2%	466	447	847 (44.43%)	39.5%	23	19
Italian	11/2008	7,726	5,175 (67.0%)	136 (2.6%)	78.0%	42	42	112 (82.3%)	91.0%	7	6
Total		356,931	341,803 (96%)	142,513 (42%)	95%	3,186	2,969	60,032 (42%)	96.1%	76	49

Table 3: Datasets used.

5. EVALUATION

This section describes our datasets and the results of analyzing them through our infrastructure.

5.1 Datasets

Table 3 details the datasets used. The first two columns show the name of the dataset and its release date. Our main source of samples is VirusShare [14] from where we download 5 datasets between 2012 and February 2015. We also obtain from Italian collaborators a dataset of unlabeled older samples and collect a small dataset of samples with encrypted network traffic. We include the publicly available Malicia dataset [35], which contains labeled samples that we use to evaluate our clustering. The last dataset contains samples downloaded from the CCSS Forum certificate blacklist [5] used for measuring our coverage.

The next two columns summarizes the executables in the dataset. First, it shows the number of PE executables in the dataset, after excluding other malicious files (e.g., HTML). Overall, our infrastructure processed 356,931 executables. Then, it shows the number and percentage of malicious and potentially unwanted executables in the dataset. An executable is malicious or unwanted if more than 3 AV engines flag it in VirusTotal [15]. As expected, the vast majority (96%) satisfy this condition.

The next group of 4 columns (*CS chain*) summarizes the signed executables and their code signing chains. It shows the number of signed malicious executables, the fraction of signed samples classified as PUP using the cluster classification, the number of unique certificate chains in those executables, and the number of distinct leaf certificates in those chains. Overall, 142,513 samples (42%) are signed of which 95% are PUP and 5% malware. Those signed samples contain 3,186 distinct chains. On average, 45 signed samples share the same certificate chain, which is an indication that they belong to the same operation. We detail the clustering into operations and PUP classification in Section 5.2. Those 3,186 chains contain 2,969 unique leaf (i.e., code signing) certificates.

The last group of 4 columns (*TS chain*) summarizes the timestamped executables, and their timestamping chains. It shows the number and percentage of timestamped malware over all signed malware, the fraction of those samples classified as PUP, the number of unique timestamping certificate chains in those executables, and the number of distinct leaf certificates in those timestamping chains. Overall, 42% of the signed samples are also timestamped. Those files contain only 76 distinct chains, with 49 unique leaf certificates. On average, 790 samples share the same timestamping chain, a significantly larger reuse compared to code signing chains indicating that TSA infrastructure is quite stable. Oftentimes, executables are signed by one CA and timestamped by a different CA.

Dataset collection. We know that the Malicia dataset was collected from drive-by downloads, which silently install malware on victim computers. Silent installs are characteristic of malware while PUP tends to be distributed through bundles or installers. Thus, the Malicia dataset is biased towards malware. In fact, given the available labels we know that it contains no PUP (neither signed or unsigned). Unfortunately, we do not know how datasets other than Malicia were collected, a common situation with third-party datasets. In particular we do not know whether the VirusShare datasets, which are the largest and dominate our corpus, may have some bias towards PUP or malware due to their collection methods. We leave as future work replicating the analysis in other datasets to compensate for any possible collection bias in VirusShare.

Samples	**Clusters**	**Singletons**	**Largest**	**Mean**	**Median**
142,513	2,288	1,432	42,711	62.3	1

Table 4: Clustering results on signed samples.

5.2 Clustering and PUP Classification

Table 4 summarizes the clustering results on the 142,513 signed samples. The clustering outputs 2,288 clusters of which 1,432 contain only one sample. The distribution is skewed with the largest cluster containing 42,711 samples, the average cluster 62.3, but the median only one due to the large number of singletons. We detail the top operations in Section 5.7. To evaluate the clustering accuracy we manually analyze the 235 clusters with more than 10 samples, which cover 97% of signed samples. We observe high precision but lower recall, i.e., some operations are split into multiple clusters typically one large cluster and one or two small clusters. This is consistent with the ground truth evaluation in Table 2.

PUP cluster classification. Our PUP cluster classification applied on the 2,288 clusters of signed samples outputs that 721 clusters are PUP and 1,567 malware. While a majority of clusters are labeled as malware, the largest clusters are labeled as PUP and overall the cluster classification considers 95% of the samples as PUP and 5% as malware. The median PUP cluster size is 188 samples and for malware clusters 4.4 samples. Over the top 235 clusters manually examined, we find 10 where our manual PUP classification differs from the automatic classification. The largest of these 10 clusters has 351 samples and altogether they comprise 890 potentially misclassified samples, 0.64% of all manually labeled samples.

PUP sample classification. The PUP sample classification is applied to 341,119 signed and unsigned samples, for which we have a VT report and they are detected by at least one of the selected 11 AV engines. Of those, 44% are labeled PUP and 56% malware. This indicates that our corpus is quite balanced on malware and PUP samples. For signed samples, 88% are labeled PUP and 12% malware. For unsigned samples, the results are almost opposite: 11% are labeled PUP and 89% malware. These numbers indicate

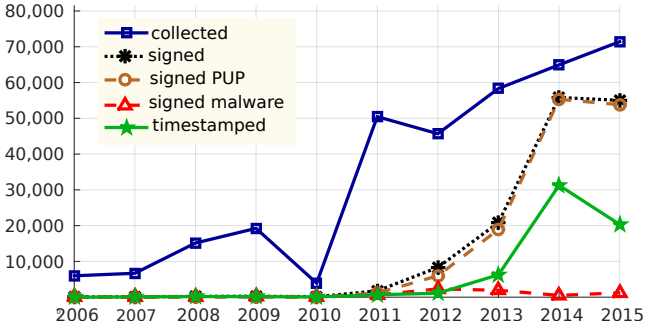

Figure 4: Number of collected, signed, timestamped, signed PUP, and signed malware samples over time. The cluster classification is used to label signed PUP and malware samples.

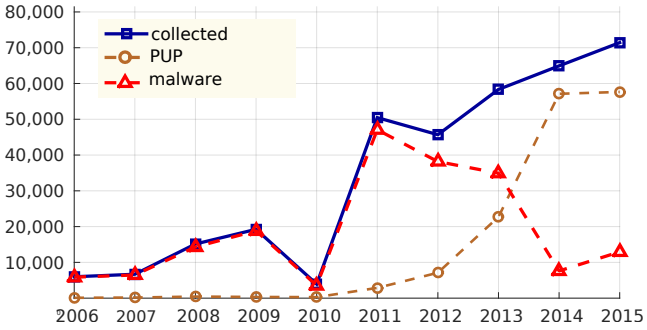

Figure 5: Number of collected, PUP, and malware samples over time including both signed and unsigned samples. The sample classification is used to label PUP and malware samples.

that PUP is most often signed, but malware only rarely, an important conclusion of our work.

Table 5 summarizes the PUP classification. As expected, the cluster classification labels as PUP more signed samples (95% versus 88%) since it considers as PUP samples that may not be individually labeled as PUP, but belong to a PUP dominated cluster. Our manual analysis of samples with differing classification observes a higher accuracy for the cluster classification. When not mentioned explicitly throughout the evaluation, the PUP classification results are those of the cluster classification.

5.3 Evolution over Time

We analyze if malware and PUP are increasingly being signed. To examine the evolution over time, we need to approximate when samples were created. The majority of dates embedded in executables (e.g., compilation time) are unauthenticated and can be forged. The timestamping date is authenticated, but only available in 42% of the signed samples. Thus, we approximate the creation date of each sample by the first submission to VirusTotal.

Figure 4 plots for each year between 2006 and 2015 the number of collected samples (signed and unsigned) in our corpus first seen by VT on that year, as well as the number of signed samples, signed PUP, signed malware, and timestamped samples (both malware and PUP). PUP and malware signed samples are labeled using the cluster classification. The figure shows that signed samples were rare in our corpus until 2011. Since then, they have steadily risen with the majority (87%) of all samples collected in 2014 being signed. This growth is due to the increase of signed PUP, as the number of signed malware has kept steadily low over time.

	Signed		Unsigned		All samples	
Classification	**PUP**	**Mal.**	**PUP**	**Mal.**	**PUP**	**Mal.**
Per Sample	88%	12%	11%	89%	44%	56%
Per Cluster	95%	5%	-	-	-	-

Table 5: Summary of PUP classification results.

Validation Result	**Signed Files**	**PUP**	**Mal.**
OK	95,277 (66.8%)	69.2%	21.7%
CERT_E_REVOKED	23,550 (16.5%)	16.9%	9.7%
CERT_E_EXPIRED	19,016 (13.3%)	13.7%	5.4%
TRUST_E_BAD_DIGEST	2,798 (2.0%)	<0.1%	38.3%
CERT_E_UNTRUSTEDROOT	1,136 (0.8%)	<0.1%	15.9%
TRUST_E_NOSIGNATURE	503 (0.3%)	<0.1%	7.0%
CERT_E_CHAINING	170 (0.1%)	<0.1%	1.0%
CERT_E_UNTRUSTEDTESTROOT	47 (<0.1%)	<0.1%	0.6%
TRUST_E_COUNTER_SIGNER	8 (<0.1%)	0%	0.1%
TRUST_E_NO_SIGNER_CERT	7 (<0.1%)	0%	0.1%
CERT_E_WRONG_USAGE	1 (<0.1%)	0%	<0.1%
Total	142,513 (100%)	100%	100%

Table 6: Validation results using the default Windows policy.

The figure also shows the increase of timestamped samples over time, which starts in 2012 and rises more slowly, achieving 49% of all collected samples being timestamped in 2014. Note that the dip in 2010 is due to our corpus, not to less malware and PUP having been produced that year. In general, malware and PUP have been growing steadily over the years [29]. The dips in 2015 happen because only January and February are included.

Figure 5 is similar but it includes both signed and unsigned samples, labeled using the sample classification. It shows that in our corpus PUP has been increasing over time and the increase in PUP highly resembles the signed PUP increase in Figure 4, despite using different PUP classification metrics. In contrast, malware has been decreasing in our corpus since 2011. This could indicate that PUP is replacing malware over time, but could also be due to collection bias on the VirusShare datasets. We leave as future work examining this trend on other datasets.

5.4 Authenticode Validation

All signed samples are validated using the default Windows Authenticode policy. Table 6 summarizes the validation results. The majority (67%) of signed samples still validates correctly in Windows. The remaining 33% fail Windows Authenticode validation. The most common validation error is that a certificate has been revoked (*CERT_E_REVOKED*) returned for 16.5% of the signed samples. The second most common validation error is that a certificate in the chain has expired (*CERT_E_EXPIRED*), which affects 13.3% of signed samples.

Note that revoked and expired code signing certificates were valid when they were issued. Thus, the total number of signed samples that used a CA-issued certificate is 97%. And, 73% of leaf certificates used to sign the samples have been issued by CAs. The other are self-signed or bogus.

The two rightmost columns in Table 6 show the percentage of Authenticode validation results for PUP and malware respectively. Only 22% of signed malware still validates, compared to 69% of PUP. When including revoked and expired certificates we observe that 99.8% of PUP samples had at some point a valid signature, compared to 37% of malware. Thus, PUP authors have no trouble obtaining valid certificates from CAs. For malware authors, identity checks by CAs seems to present a higher barrier. Still, over one third of the signed malware had at some point a valid signature. The fact that less malware samples are revoked compared to PUP

	Issued			Revoked			Hard Revocations		
CA	**Total**	**PUP**	**Malware**	**Total**	**PUP**	**Malware**	**Total**	**PUP**	**Malware**
Symantec/VeriSign	708	70.5%	29.5%	76 (10.7%)	7.2%	19.1%	23 (30.2%)	44.4%	17.5%
Symantec/Thawte	510	66.0%	34.0%	109 (21.4%)	24.6%	15.0%	4 (3.7%)	2.4%	7.7%
Comodo	406	85.0%	15.0%	60 (14.8%)	15.4%	11.5%	54 (90.0%)	88.7%	100%
GlobalSign	153	80.0%	20.0%	14 (9.1%)	9.8%	6.4%	0	0%	0%
WoSign	120	35.8%	64.2%	10 (8.3%)	7.0%	9.0%	7 (70%)	66.6%	71.4%
GoDaddy/StarField	99	85.0%	15.0%	28 (28.3%)	31.0%	13.3%	6 (21.4%)	23.0%	0%
DigiCert	85	68.2%	31.8%	37 (43.5%)	36.2%	59.2%	9 (24.3%)	14.3%	37.5%
Certum	32	65.6%	34.4%	7 (21.9%)	14.3%	36.4%	0	0%	0%
Symantec	23	87.0%	13.0%	0	0%	0%	0	0%	0%
StartCom/StartSSL	10	60.0%	40.0%	2 (20%)	16.6%	25%	0	0%	0%
Total	2,170	71.0%	29.0%	343 (15.8%)	15.4%	16.7%	103 (30.0%)	32.0%	25.7%

Table 7: Leaf certificates issued and revoked by CAs and used to sign PUP and malware.

samples is due to PUP authors reusing certificates to sign a larger number of samples than malware authors. Certificate revocations are similar for both classes and detailed in Section 5.5.

The vast majority of other validation errors are due to malware. A significant (2.0%) fraction of samples have digital signatures that cannot be verified (*TRUST_E_BAD_DIGEST*) because the Authentihash in the PKCS7 structure does not match the file's Authentihash. This is the most common Authenticode validation result for malware samples and is often due to malware authors copying certificate chains from benign executables onto their malware. For example, the most common Subject CN of these leaf certificates is for Microsoft Corporation. Copying a benign certificate chain on a malware sample changes the sample's file hash and also invalidates byte signatures on the certificates themselves, without changing the malware code. This may help to bypass some AV engines and explain why we observe multiple malware samples with the same Authentihash, but different certificate chains.

Another popular validation error among malware is an untrusted root certificate (*CERT_E_UNTRUSTEDROOT*) not included in the default Windows trust store. The majority of these (1,102/1,136) contain chains with only one self-signed certificate. Another 34 contain fake certificates for valid CAs and the rest are bogus.

There are 503 samples (491 malware) that Windows does not consider signed (*TRUST_E_NOSIGNATURE*). These contain misplaced Authenticode signatures, which Windows does not identify but our parsing code does. Another 170 (73 malware) samples contain chains where the certificates are not in the proper order (*CERT_E_CHAINING*), a phenomenon also observed in SSL certificate chains [22].

There are 47 samples whose chains end with a root certificate created by Microsoft's Certificate Creation Tool[2], used by developers to test code under development (*CERT_E_UNTRUSTEDTESTROOT*). Eight samples contain an invalid timestamping chain (*TRUST_E_COUNTER_SIGNER*). For seven samples Windows is not able to find the leaf certificate (*TRUST_E_NO_SIGNER_CERT*). The final sample contains a leaf certificate without the code signing flag (*CERT_E_WRONG_USAGE*).

5.5 Revocation

In this section we examine the revocation of certificates used to sign PUP and malware. For this, we use OCSP and CRL revocation checks that our infrastructure performs for each certificate using OpenSSL. We do not use the *CERT_E_REVOKED* Authenticode validation error because it does not specify which certificate in the chain was revoked and because other errors may hide the revocation [20]. Of the 2,969 leaf certificates, 83% contain a CRL URL, 78% both CRL and OCSP URLs, and 17% neither[3]. Revocation checks are successful for 90% of the certificates with a CRL or OCSP URL, the remaining 10% fail. The most common errors are OCSP unauthorized (i.e., CA does not recognize the certificate typically because it is fake) and an empty CRL list.

Table 7 summarizes the code signing certificates issued by each CA and used to sign PUP or malware in our corpus, and their revocations. For each CA it shows the number of valid certificates issued (including those that still validate, have been revoked, and have expired but were valid otherwise), the number of certificates revoked, and the number of hard revocations performed by the CA. It also provides the split of those categories into certificates that sign PUP and malware respectively.

Overall, 2,170 out of 2,969 leaf certificates were issued by CAs, the rest are self-signed or bogus. Symantec's Verisign and Thawte brands issue most code signing certificates used to sign PUP and malware. This may be due to Symantec having the largest market share of the code signing market. Unfortunately, we did not find any public code signing CA market share figures to compare with. Of those 2,170 certificates, 71% are used to sign PUP and 29% malware. All CAs issue more certificates to PUP authors except WoSign, a Chinese CA. These results indicate that obtaining a CA-issued code signing certificate may be easier for PUP authors, but malware authors still often manage to obtain one.

All revocations are for leaf certificates. Overall, 343 code signing certificates have been revoked. Thus, CAs revoke less than 16% of the certificates they issue to PUP and malware authors. The PUP and malware percentages are computed over the number of certificates issued to PUP and malware authors, respectively. There is no significant difference in the percentage of PUP certificates that gets revoked (15.4%) compared to malware certificates (16.7%). Five CAs revoke a higher percentage of malware certificates and 4 a higher percentage of PUP certificates. Both results indicate that CAs revoke similarly certificates used by PUP and malware.

Thawte is the CA with most revoked certificates and DigiCert the CA revoking the largest fraction of malicious certificates it issued. No CA revokes more than 43% of their abused certificates. These numbers indicate that revocation is currently not an effective defense against abused code signing certificates. We further discuss this at the end of this subsection.

The average time to revoke a certificate is 133 days. Comodo is the fastest to revoke malicious certificates (21 days) although it only revokes 15% of them. Verisign is significantly slower (validity > 9

[2] `https://msdn.microsoft.com/en-us/library/bfsktky3.aspx`

[3] One leaf certificate contains only OCSP URL.

	Leaf Certificates			
Reason	**All**	**PUP**	**Malware**	**# CA**
Unspecified / NULL	163 (47%)	67.5%	32.5%	7
Key Compromise	137 (40%)	69.3%	30.7%	2
Cessation of Operation	35 (10%)	80.0%	20.0%	3
Superseded	6 (2%)	50.0%	50.0%	2
Affiliation Changed	2 (<1%)	100%	0%	2

Table 8: Summary of revocation reasons.

months) than the other CAs to revoke malware-used code signing certificates and only revokes 11%.

All revocations are available through OCSP and only a handful through CRLs. The reason may be that expired certificates are removed from CRLs to prevent them from growing too large, a behavior allowed by RFC 2459 [26]. We find some revocations for GoDaddy/Starfield that appear in CRLs but not through OCSP. This inconsistency indicates the need to check both revocation methods for this provider.

The vast majority (96.2%) of revocations happen during a certificate's validity period. We only observe 13 certificates revoked after they have expired. A revocation after expiration has no effect in Windows validation.

Revocation reason. A revocation may optionally include a revocation reason [21, 38]. Table 8 details the revocation reasons returned by OCSP or in the CRL. The reason is unspecified or not provided at all in 47% of revocations. The most common revocation reason is key compromise used in 40% of revocations by two CAs: Thawte and VeriSign. The key compromise reason is used not only in cases where the certificate's owner may have reported a key compromise but also when the CAs were likely deceived to issue a certificate to a malicious publisher. For example, 30% of these certificates were issued to malware publishers, which are unlikely to report a key compromise. It seems that CAs do not care about giving precise revocation reasons and this field is currently not useful.

Hard revocations. We observe some CAs (WoSign, Comodo, VeriSign, GoDaddy, DigiCert, Thawte) performing some revocations on the certificate issue date. This could indicate that they are already performing hard revocations or that they want to hide when they discovered the certificate's abuse. We have not found any prior references on the need or use of hard revocations. Comodo (90%) and WoSign (70%) have the highest fraction of such revocations. Unfortunately, they never provide a revocation reason. Our analysis of these revocations reveals that they are not performed systematically. For example, WoSign revokes two certificates from the same operation, with the same Subject CN and one gets a revocation on the issue date and the other does not.

Summary of findings. Our revocation analysis shows that less than 16% of CA-issued code signing certificates used by malware are revoked with no significant difference between certificates used by malware (17% revoked) and PUP (15%). The lack of revocation is widespread across CAs: no CA revokes over 43% of the abused code signing certificates it issued. In addition, CAs do not properly detail the reason for which a certificate was revoked, which makes it difficult to separate key compromises from certificates purposefully obtained to sign malware and PUP. Some CAs perform revocations on the issue date. They may have realized the need of hard revocations. But, we have not seen any references to this issue, most CAs seem unaware, and the ones performing them show inconsistencies in their use. These findings support that revocation of malicious code signing certificates is currently ineffective.

[4]Includes TC TrustCenter GmbH, acquired by Symantec

CA	**Samples**	**Chains**
Symantec/VeriSign[4]	43,295 (72%)	12
GlobalSign	13,536 (22%)	5
Comodo	1,878 (3%)	8
DigiCert	630 (<1%)	7
GoDaddy/Starfield	316 (<1%)	3
WoSign	174 (<1%)	7
Entrust	42 (<1%)	5
Microsoft	126 (<1%)	21
Certum	20 (<1%)	2
Yessign	3 (<1%)	2
Daemon Tools	2 (<1%)	2
GeoTrust	2 (<1%)	1

Table 9: Timestamping authorities used by malware and PUP: number of samples and timestamping chains for each TSA.

5.6 Timestamping

We have already shown (Table 3) that 42% of the signed samples in our corpus are timestamped and that timestamped samples are on the rise (Section 5.3). In this section we detail the usage of timestamping by PUP and malware. Table 9 shows the timestamp authorities (TSA) used by samples in our corpus. For each TSA, the table presents the number of samples that were timestamped by this TSA and the number of distinct timestamping certificates chains for the TSA.

The results show that Symantec/Verisign is the most popular TSA, used by 72% of the timestamped samples, followed by GlobalSign with 22%. Next, we show that TSAs do not perform checks on executables sent to be timestamped. Thus, the popularity of Symantec's and GlobalSign's TSAs among PUP and malware authors is not due to these providers performing less validation than other TSAs, but most likely due to a larger market share. Note that Microsoft and Daemon Tools are not publicly available TSAs, some authors copied the timestamping chains from other files into their executables. These samples do not validate.

Lack of timestamping checks. We perform an experiment to test whether TSAs perform any checks on executables they receive for timestamping. We select 22 signed samples from our corpus, two for each Authenticode validation result in Table 6. We use the Windows SignTool [11] to send those samples to the top 7 TSAs in Table 9. All 7 TSAs successfully timestamped 20 of the 22 samples. The only two samples that were not timestamped were those with Authenticode validation error TRUST_E_NO_SIGNATURE (Section 5.4). Those samples have their signatures in a wrong position. We also try timestamping an already timestamped file, which results in replacement of the old timestamp with a new one. In summary, we do not observe any restrictions imposed by TSAs on the executables to be timestamped, other than they should be signed. TSAs do not check that the executable's certificate chain validates and do not attempt to identify malicious or potentially unwanted software. Given that timestamping is a free service, TSAs may not have an incentive to invest in checks.

Timestamped and revoked. Timestamping is beneficial for authors since if a sample is timestamped before its code signing certificate is revoked, then Windows authentication will always succeed on that sample, regardless of the revocation. In our corpus we find 911 timestamped samples with a revoked certificate. A total of 118 revoked code signing certificates are used by these samples. The low number of samples in this category is due to less than 16% of abused code signing certificates being revoked. Of those samples, 655 (72%) are timestamped before their code signing certificate is revoked. These samples will continue to successfully val-

		Dates		Samples		Certificates					Certificate Subjects				
Name	**Type**	**Certificates**	**Malware**	**Signed**	**TimeSt.**	**Issued**	**Revoked**	**Hard Rev.**	**Avg. Validity**	**CAs**	**CNs**	**Comp.**	**Ind.**	**CCs**	**Cost**
Firseria	PUP	05/11 - 09/17	08/11 - 02/15	42,711	42,543	26	0	0	1.7	5	20	15	0	2	$12,734
SoftPulse	PUP	02/14 - 01/16	07/14 - 02/15	21,083	1	43	4	0	1.0	6	20	13	0	2	$15,959
InstallRex	PUP	03/11 - 07/16	10/11 - 02/15	12,574	0	51	21	20	1.1	3	45	2	43	4	$10,394
Tuguu	PUP	05/12 - 06/15	01/13 - 02/15	7,891	3	34	22	10	1.0	6	15	7	0	4	$8,771
OutBrowse	PUP	02/13 - 08/17	07/13 - 02/15	5,590	21	97	64	0	1.0	5	44	40	0	6	$27,300
LoadMoney	PUP	12/11 - 03/16	08/12 - 02/15	5,285	38	14	9	8	1.2	2	13	12	0	1	$3,554
ClientConnect	PUP	02/12 - 12/16	06/14 - 02/15	3,576	3,562	21	0	0	2.0	3	3	3	0	3	$17,760
InstallCore	PUP	07/10 - 01/17	01/11 - 02/15	2,972	900	101	3	2	1.2	6	89	75	0	17	$29,595
Zango	PUP	05/09 - 01/15	07/10 - 09/13	2,913	25	6	5	5	1.9	1	3	3	0	1	$4,864
Bundlore	PUP	07/11 - 07/16	12/12 - 02/15	2,823	0	6	0	0	1.5	2	3	2	0	1	$1,797

Table 10: Top 10 operations. The validity period is in years and the cost in US Dollars.

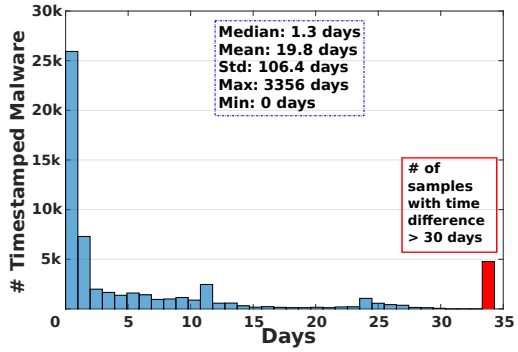

Figure 6: Time difference in days between a sample was timestamped and it was first observed in VirusTotal. There are 44 samples with a negative time difference of at most -10 minutes that are not shown in the figure.

idate after revocation. The remaining 28% are timestamped after revocation, up to 5.6 months after their code signing certificate was revoked. Thus, some authors keep using their code signing certificate long after it has been revoked. They still see value in signing their executables even when the signature does not validate, or did not realize that the revocation happened.

Timestamping speed. Next, we examine whether timestamping happens close to the creation time of a sample. For this we compare the timestamping date with the first time the timestamped sample was observed by VirusTotal (VT). As expected, the vast majority of samples are observed by VT after the timestamping date. Out of 60 K timestamped samples, only 44 are observed by VirusTotal before they are timestamped, and all those are seen by VT within 10 minutes of the timestamping date. These 44 samples are likely sent to VT to check if they are detected by AVs before timestamping them. This indicates that timestamping happens closely after a sample is signed and before it starts being distributed. Otherwise, we would expect VT to see a larger number of samples distributed before timestamping and over a larger time frame. The consequence of this is that the timestamping date is a highly accurate estimation of the creation time. This is important because typically we have no reliable indication of when a sample is created. In practice, many works use the first-seen-on-the-wild date as an approximation.

We can use the timestamping date to evaluate how fast malware repositories collect samples, something that we are not aware has been measured earlier. Figure 6 shows the time difference in days between a sample was timestamped and it was first observed in VT. Overall, it takes VT a median of 1.3 days to observe a sample, but the distribution is long-tailed. The red bar on the right of the figure shows that 8% of the timestamped samples are seen by VT over a month after they are created. This happens more often with older samples created while VT did not have as good coverage as it does now. In the worst case, some samples are seen by VT more than 6 years after they were created. Thus, using the first-seen-on-the-wild date as an approximation of creation time for a sample works for the majority of recent samples, but can introduce large errors with a small percentage (<8%). Using the timestamping date is a more accurate estimation that does not rely on the distribution channel. While only 42% of our samples are timestamped, we have shown that timestamping is growing.

5.7 Largest Operations

In this section we use the clustering results to analyze the code signing infrastructure of the largest operations in our corpus. When sorting the clusters in Table 4 by number of signed samples they contain, the top 21 clusters correspond to PUP operations. The first malware cluster at rank 22 corresponds to Zbot. However, aggregating all Zbot clusters would rank Zbot as 11th largest operation. When we sort clusters by the number of CA-issued leaf certificates the first malware cluster has rank 22 and uses 7 certificates.

Table 10 summarizes the top 10 operations, all PUP, in decreasing order of signed samples. The left half of the table shows, for each operation, the operation name, whether it corresponds to PUP or malware, the number of signed and timestamped samples, the number of certificates issued and revoked, and the average validity period in years of all certificates issued to the operation. The right half of the table details the subjects of the certificates issued to the operation, the number of CAs that issued those certificates, and the estimated certificate cost for the operation in US dollars.

File polymorphism. The 10 PUP operations in Table 10 distribute 75% of the signed samples in our corpus. The top operation (Firseria) distributes 30% of the signed samples alone, and the top 3 more than half. Thus, large PUP operations heavily use file polymorphism. For example, SoftPulse produces at least 21 K signed samples in 7 months, an average of 97 new signed samples per day. Such polymorphism is likely used to avoid AV detection and is a behavior often associated with malware.

Two of the top 10 families (Firseria and ClientConnect) timestamp the vast majority of their signed samples. Thus, some PUP authors have already realized the benefits of timestamping. The rest have no timestamped samples, or only a handful likely due to tests or third-party timestamping (like we did in Section 5.6).

Certificates. These 10 operations use from 6 code signing certificates (Zango, Bundlore) up to 84 certificates (OutBrowse). On average, they sign 445 samples with the same code signing certificate, amortizing the certificate cost over many samples. The average lifetime of their certificates ranges from one year for 3 operations to

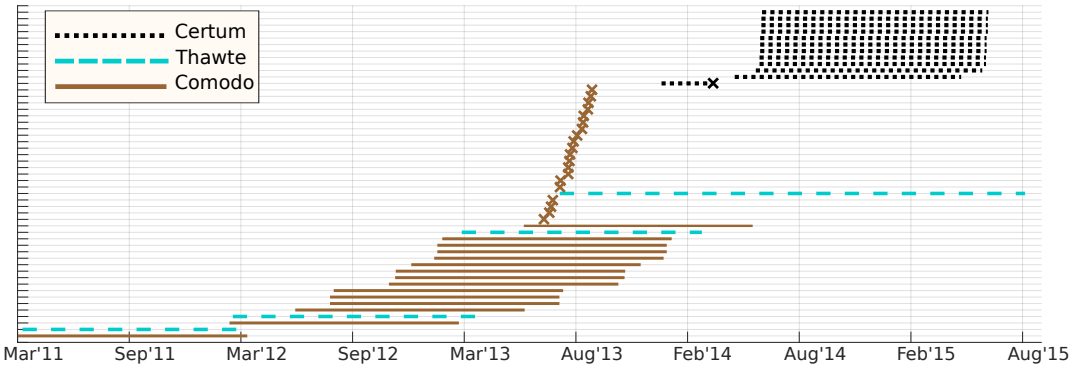

Figure 7: CA-issued certificates used by the InstallRex operation over time. Each line corresponds to a different certificate and its length marks the period between the certificate issuing date and its expiration or revocation (denoted by a cross) date. A single cross in one line indicates a hard revocation, i.e., a revocation on the certificate issuing date.

two years for 2 operations. Three of the operations favor 2-year certificates (validity larger than 1.5) and 6 favor 1-year certificates (validity less than 1.5). The longer the validity period the larger the investment loss if a certificate gets revoked.

Certificate revocations. Seven of the 10 families have multiple certificates revoked. It seems unreasonable that an entity would have 3–61 key compromises, so those revocations are likely due to malicious behavior. This indicates that CAs consider those 7 PUP operations malicious. For operations with revoked certificates, revocation does not work great since at most 66% of their certificates (OutBrowse) are revoked.

Interestingly, the two operations that timestamp their files do not have revocations and the 3 operations with zero revocations (Firseria, ClientConnect, and Bundlore) favor 2-year certificates. Their lack of revocations seems to give them enough confidence to commit to larger investments. Additionally, buying longer-lived certificates makes them look more benign, further contributing to the lack of revocations.

Certificate polymorphism. Eight of the 10 operations use over 10 code signing certificates and 9 buy certificates from multiple CAs. The right part of Table 10 examines who requested the code signing certificates (i.e., the certificate Subject field). First, it shows the number of distinct Subject CN fields in the certificates, then the grouping of those into unique companies or individuals that requested the certificates, and finally the number of countries for those subjects. These 10 operations use 399 certificates with 255 distinct Subject CNs. On average, 1.6 certificates have the same Subject CN. After grouping similar Subject CNs, (e.g., "Tuguu SL" and "Tuguu S.L.U.") those 399 certificates correspond to 172 corporations and 43 individuals. All individual certificates are used by the InstallRex operation. The other operations use corporations to buy the certificates. Five of the operations use more than 10 corporations. For some operations (e.g., Tuguu) we are able to check the company information on public business registers showing that the same person is behind multiple companies used by the operation. For each operation, the corporations and individuals are concentrated in a few countries, most often the United States and Israel.

These results show that PUP operations heavily rely on certificate polymorphism through the use of multiple CAs, small modifications of Subject CNs, and buying the certificates through multiple corporations or individuals. Such certificate polymorphism is likely used to bypass CA identity validation and revocation, increasing the resilience of their certificate infrastructure. For example, Comodo revokes a LoadMoney certificate issued for *LLC Monitor* but the family possess another one from Thawte issued for *Monitor LLC*, which due to the lack of CA synchronization is not revoked. Overall, operations have adapted to obtain new certificates when their current ones are revoked. We show an example for the InstallRex operation at the end of this subsection.

Cost. We estimate the cost of the certificate infrastructure for these operations by adding the cost of all certificates issued to the operation using the per CA and per validity period certificate costs in our market analysis. Certificate prices may have changed over the years and we may only have an incomplete view of the certificates used. Still, we believe this estimate provides a good relative ranking. The investment on code signing certificates by these operations varies from $1,797 (Bundlore) to $29,595 (InstallCore) with an average of $13,272.

InstallRex. Figure 7 shows the certificates of the InstallRex operation over time. Each line corresponds to a certificate's validity period. Crosses mark revocation dates. A single cross in a line indicates the CA performed a revocation on the issue date. InstallRex uses 51 certificates from 3 CAs. From March 2011 until April 2013 they bought 14 personal certificates from Comodo using different identities and one personal and another for a company ("Web Pick - Internet Holdings Ltd") from Thawte. Starting on June 2013 they acquired 22 personal certificates from Comodo and another from Thawte for the same company and different capitalization ("WEB PICK - INTERNET HOLDINGS LTD"). This time Comodo realized and issued revocations on the expiration dates for all their certificates, but Thawte did not revoke theirs. A few months later they start acquiring personal certificates from Certum. The first one is revoked after some months, but a month later they succeed to buy 11 different certificates from Certum, which have not been revoked. This example illustrates how PUP operations exploit the lack of CA synchronization and multiple identities to survive revocations.

5.8 Blacklist Coverage

The certificate blacklist output by our infrastructure contains 2,170 CA-issued code signing certificates. In comparison, the CCSS blacklist [5] contained on May 2015 entries for 228 code signing certificates. Of those, only 197 provided a VirusTotal link to a malware sample signed with that certificate. We analyzed those 197 samples. Three of them were not considered malicious using our rule, another 19 are not really signed (according to both our infrastructure and VT), and 3 share certificate. Overall, our blacklist contains 9x more certificates. We further discuss blacklist coverage in Section 6.

6. DISCUSSION

Hard revocation deployment. Hard revocations can be used without any changes to the Authenticode implementation in Windows. However, Microsoft could support its deployment by communicating to CAs both type of revocations and the recommended handling of key compromises and certificate abuse. One straightforward way to achieve this would be updating the 2008 Authenticode specification [31]. CAs can already use hard revocations, but it is important that they provide abuse email addresses to receive third-party notifications of abuse.

PUP maliciousness. PUP has been understudied in the research literature. Important questions such as the (lack of) behaviors that make a program PUP rather than malware remain open. This makes it possible for malware to disguise as PUP. We do not attempt to define what constitutes PUP, but rather rely on AV labels for that determination. However, our work is an important first step towards understanding PUP. We observe that PUP may be quickly growing and that it is typically signed. We also observe many PUP operations with suspicious behaviors such as high file and certificate polymorphism that could also be associated with malware.

Identity checks. CAs should implement checks to avoid identities to be reused with slight modifications. They should also provide correct revocation reasons to enable distinguishing revocations due to key compromise and abuse, which is important to build publisher reputation. Log-based PKI solutions [28] where CAs submit all their issued certificates to a central repository would help identifying identity reuse across CAs.

Blacklists. Certificate blacklists would not be needed if revocation worked properly. However, they are an important stopgap solution and may help pushing CAs to improve revocation. We have shown that automatic approaches can build blacklists with an order of magnitude larger coverage than existing ones. To achieve larger coverage it is important that AV vendors and malware repositories contribute signed malware or their certificates to existing blacklists.

7. RELATED WORK

Code signing. Code signing is a key component of binary integrity solutions. DigSig [17] presents a Linux kernel module that validates digital signatures of programs before execution. Wurster and van Oorschot [43] protect executables from malicious modifications using self-signed certificates, where the OS kernel allows modifications only if the current and the new version of the file are signed with the same private key. Wu and Yap [42] leverage code signing in their binary integrity model. Application whitelisting relies on code signing to obtain publisher identity [23]. Recent work has examined the challenges of transparent key updates and certificate renewals in Android applications [19].

Attacks on Authenticode. Prior work has shown the possibility of injecting code and data into Authenticode signed executables without invalidating the signature [25, 30] and that executables signed using MD5 are vulnerable to collisions [39].

Authenticode measurements. Most similar to our work are measurements of signed malware by two AV vendors in 2010 [37, 41]. Those works focus on 2008-2010, while our analysis covers an 8-year span (2006-2015). Our span covers the significant increase in malware code signing after 2010. Our analysis covers many aspects not addressed in those studies such as timestamping. Our infrastructure clusters samples into operations and classifies each of them as PUP or malware, enabling the analysis of specific operations. We also analyze the revocation of timestamped executables and show the need for hard revocations.

HTTPS certificates. Prior work analyzes the HTTPS certificate ecosystem revealing many bad practices [22] and flaws in the certificate validation process [20, 24]. Our work shows that Authenticode validation is as complex or more compared to SSL/TLS validation (e.g., includes timestamping) and since it's proprietary there is a need for further external evaluation of its security.

8. CONCLUSION

We have performed a systematic analysis of Windows Authenticode code signing abuse and the effectiveness of CA defenses. We have identified a problematic scenario in Authenticode where timestamped signed malware successfully validates even after the revocation of their code signing certificate. We have proposed hard revocations as a solution. We have built an infrastructure that automatically analyzes potentially malicious samples, filters out benign and unsigned samples, clusters the remaining into operations, classifies them as PUP or malware. At last, it produces a blacklist of malicious certificates.

We have evaluated our infrastructure on 356 K samples and observe that PUP is rapidly increasing in our corpus, that most PUP is signed, and that signed malware is not prevalent. CA identity checks pose some barrier to malware (37% signed malware use a CA-issued certificate) but do not affect PUP (99.8%). Revocation is also limited as only 17% of malware certificates and 15% of PUP certificates in our corpus have been revoked. We analyze the largest PUP operations showing that they heavily use file and certificate polymorphism. They buy certificates from multiple CAs, apply small modifications to certificate subjects to reuse identities, and use multiple companies and individuals to buy the certificates. We have also used timestamped malware to evaluate the speed with which the VirusTotal online service collects malware.

9. ACKNOWLEDGMENTS

We are grateful to VirusTotal and VirusShare for making their data publicly available. We thank Thorsten Holz and the anonymous reviewers for their insightful comments and feedback.

This research was partially supported by the Regional Government of Madrid through the N-GREENS Software-CM project S2013/ICE-2731 and by the Spanish Government through the StrongSoft Grant TIN2012-39391-C04-01. All opinions, findings and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

10. REFERENCES

[1] Allowing only signed application to run. https://technet.microsoft.com/en-us/library/dd723683\%28v=ws.10\%29.aspx.

[2] Ca security council. https://casecurity.org/.

[3] Ca/browser forum. https://cabforum.org/.

[4] Catalog files and digital signatures. https://msdn.microsoft.com/en-us/library/windows/hardware/ff537872\%28v=vs.85\%29.aspx.

[5] Ccss forum: Common computing security standards. http://www.ccssforum.org/.

[6] Cross-certificates for kernel mode code signing. https://msdn.microsoft.com/en-us/library/windows/hardware/dn170454\%28v=vs.85\%29.aspx.

[7] Malsign Project. http://www.malsign.org/.

[8] Malware Analysis Report - W64/Regin, Stage 1. https://www.f-secure.com/documents/996508/1030745/w64_regin_stage_1.pdf.

[9] Malwarebytes PUP Reconsideration Information. https://www.malwarebytes.org/pup/.

[10] Practical windows code and driver signing. http://www.davidegrayson.com/signing/.

[11] Signtool. https://msdn.microsoft.com/en-us/library/windows/desktop/aa387764\%28v=vs.85\%29.aspx.

[12] Stuxnet Under the Microscope. http://www.eset.com/us/resources/white-papers/Stuxnet_Under_the_Microscope.pdf.

[13] Unveiling Careto - The Masked APT. http://kasperskycontenthub.com/wp-content/uploads/sites/43/vlpdfs/unveilingthemask_v1.0.pdf.

[14] Virusshare.com repository. http://virusshare.com/.

[15] Virustotal- free online virus, malware and url scanner. http://www.virustotal.com/.

[16] Malwarebytes PUP Reconsideration Information, April 2014. http://blogs.technet.com/b/mmpc/archive/2014/04/03/adware-a-new-approach.aspx.

[17] A. Apvrille, D. Gordon, S. Hallyn, M. Pourzandi, and V. Roy. Digsig: Runtime authentication of binaries at kernel level. In *USENIX Conference on System Administration*, 2004.

[18] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *RAID*, September 2007.

[19] D. Barrera, D. McCarney, J. Clark, and P. van Oorschot. Baton: Certificate Agility for Android's Decentralized Signing Infrastructure. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2014.

[20] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov. Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations. In *IEEE Symposium on Security & Privacy*, 2014.

[21] 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 (Proposed Standard), 2008. Updated by RFC 6818.

[22] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS Certificate Ecosystem. In *ACM Internet Measurement Conference*, 2013.

[23] C. Gates, N. Li, J. Chen, and R. Proctor. CodeShield: Towards Personalized Application Whitelisting. In *Annual Computer Security Applications Conference*, 2012.

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

[25] I. Glucksmann. Injecting custom payload into signed Windows executables. In *REcon*, 2012.

[26] R. Housley, W. Ford, W. Polk, and D. Solo. Rfc 2459: Internet x. 509 public key infrastructure certificate and crl profile. 1999.

[27] B. Kaliski. Pkcs7: Cryptographic message syntax version 1.5. RFC 2315 (Proposed Standard), 1998.

[28] T. H.-J. Kim, L.-S. Huang, A. Perring, C. Jackson, and V. Gligor. Accountable key infrastructure (aki): A proposal for a public-key validation infrastructure. In *International Conference on World Wide Web*, 2013.

[29] M. Labs. Threat Report, November 2014. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2014.pdf.

[30] E. Law. Caveats for Authenticode Code Signing, September 2014. http://blogs.msdn.com/b/ieinternals/archive/2014/09/04/personalizing-installers-using-unauthenticated-data-inside-authenticode-signed-binaries.aspx.

[31] Microsoft. Windows authenticode portable executable signature format, Mar. 21 2008. http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx.

[32] A. Mohaisen and O. Alrawi. AV-Meter: An Evaluation of Antivirus Scans and Labels. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, July 2014.

[33] MSDN. Driver Signing Policy. https://msdn.microsoft.com/en-us/library/windows/hardware/ff548231.aspx.

[34] MSDN. "Stranger Danger" - Introducing SmartScreen Application Reputation. http://blogs.msdn.com/b/ie/archive/2010/10/13/stranger-danger-introducing-smartscreen-application-reputation.aspx.

[35] A. Nappa, M. Z. Rafique, and J. Caballero. The MALICIA Dataset: Identification and Analysis of Drive-by Download Operations. *International Journal of Information Security*, 14(1):15–33, February 2015.

[36] Netcraft. Keys left unchanged in many Heartbleed replacement certificates!, April 2014. http://news.netcraft.com/archives/2014/05/09/keys-left-unchanged-in-many-heartbleed-replacement-certificates.html.

[37] J. Niemala. It's signed, therefore it's clean, right?, May 2010. Presentation at the CARO 2010 Workshop.

[38] 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 (Proposed Standard), June 2013.

[39] D. Stevens. Playing with authenticode and md5 collisions, 2009. http://blog.didierstevens.com/2009/01/17/playing-with-authenticode-and-md5-collisions/.

[40] G. Wicherski. pehash: A novel approach to fast malware clustering. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[41] M. Wood. Want my autograph? The use and abuse of digital signatures by malware. In *Virus Bulletin Conference*, 2010.

[42] Y. Wu and R. H. C. Yap. Towards a Binary Integrity System for Windows. In *ACM Symposium on Information, Computer and Communications Security*, 2011.

[43] G. Wurster and P. C. van Oorschot. Self-signed Executables: Restricting Replacement of Program Binaries by Malware. In *USENIX Workshop on Hot Topics in Security*, 2007.