Artifact Evaluated NDSS Available

Understanding the Status and Strategies of the Code Signing Abuse Ecosystem

Hanqing Zhao*†, Yiming Zhang*✉, Lingyun Ying†✉, Mingming Zhang‡,
Baojun Liu*, Haixin Duan*, Zi-Quan You* and Shuhao Zhang†
*Tsinghua University, †QI-ANXIN Technology Research Institute, ‡Zhongguancun Laboratory
zhaohq23@mails.tsinghua.edu.cn, {zhangyiming,lbj,duanhx}@tsinghua.edu.cn, yinglingyun@qianxin.com,
zhangmm@mail.zgclab.edu.cn, youzq17@tsinghua.org.cn, zhangshuhao98@foxmail.com

***Abstract*—Using digital certificates to sign software is an important protection for its trustworthiness and integrity. However, attackers can abuse the mechanism to obtain signatures for malicious samples, aiding malware distribution. Despite existing work uncovering instances of code-signing abuse, the problem persists and continues to escalate. Understanding the evolution of the ecosystem and the strategies of abusers is vital to improving defense mechanisms.**

In this work, we conducted a large-scale measurement of code-signing abuse using 3,216,113 signed malicious PE files collected from the wild. Through fine-grained classification, we identified 43,286 abused certificates and categorized them into five abuse types, creating the largest labeled dataset to date. Our analysis revealed that abuse remains widespread, affecting certificates from 114 countries issued by 46 Certificate Authorities (CAs). We also observed the evolution of abuser techniques and identified current limitations in certificate revocation. Furthermore, we characterized abusers' behaviors and strategies, uncovering five tactics to evade detection, reduce costs and enhance abusing impact. Notably, we uncovered 3,484 polymorphic certificate clusters and, for the first time, documented real-world instances of malware leveraging polymorphism to evade revocation checks. Our findings expose critical flaws in current code-signing practices, and are expected to raise community awareness of the abuse threats.

I. Introduction

Code signing is a vital security mechanism that applies digitalsignatures to code or executables. It enables the verification of developer identities and software integrity. As one of the dominant operating systems, Windows adopts Authenticode [37] as its code-signing standard and offers WinTrust APIs [39] for signature verification. During installation, software from untrustworthy developers or lacking valid signatures would be detected and flagged, and alerted to users by verifiers using WinTrust APIs. Major certificate authorities (CAs) now commonly offer code-signing certificates, allowing legitimate developers with verified identities to obtain trusted certificates and use the private key to sign their software.

However, the code signing mechanism has also become a target of attacks. Attackers exploit flaws in the mechanism to obtain signatures for malware, thereby bypassing the checks of operating systems and antivirus software. This threat is known as "code-signing abuse". Notable related security breaches include Stuxnet [36] (2010), RedLine [28] (2022), and the NVIDIA certificate compromise [34] (2022), all involving stealing private keys to sign malware. Recently, VirusTotal reported [56] that nearly millions of malware samples have abused signatures, highlighting the gravity of the current situation.

So far, research on code-signing abuse remains limited in both scope and depth. Unlike the Web Public Key Infrastructure (Web PKI), where various threats have been systematically analyzed [10], [21], [32], [46], [50], the code-signing ecosystem faces unique obstacles. The lack of open, large-scale dataset (available only from real-world samples) and ground-truth on abuse cases impede scalable measurements in this field. Kim et al. [26] conducted the most comprehensive analysis of code-signing abuse to date in 2017, revealing vulnerabilities in CA issuance, client-side protections, and developer key management. Their analysis benefited from 111 certificates with specific abusing types, a scale that has yet to be expanded. As code-signing abuse remains pervasive, larger-scale fine-grained measurements are essential to have a global view of the current ecosystem, and to understand abusive behaviors and strategies that remain explored.

Research questions. Our goal is to understand the current security landscape of the code-signing abuse ecosystem, especially from the strategic level of the adversaries. Specifically, we aim to answer the following questions: *Q1: What is the current security status of the code-signing abuse ecosystem? Q2: What flaws do abusers exploit, and what strategies do they employ for abuse? Q3: What are the root causes of the rampant abuse of code signing and how to mitigate it?*

Our work. We started by building a large-scale fine-grained code signature abuse dataset for measurements. We collected 6.9M samples from VirusShare [55], spanning from Oct. 2020 to Oct. 2024, and 3.8M signed samples from a partnering security company for their inclusion from May 2006 to Sep. 2024. Particularly, we focus on code-signing for Windows portable executable (PE) files, as Authenticode is the most widely used signature mechanism and PE files are its primary target. We extracted 3,216,113 malicious PE files with code signatures as the base dataset. As 78.25% of the samples in

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.242857
www.ndss-symposium.org

our dataset were signed after 2017, our analysis shows the most recent state of the ecosystem.

To further investigate abusive behaviors and tactics, we need to understand the types of abuse and their underlying reasons. A primary challenge is the lack of ground-truth. The known largest labeled dataset of abuse types [25] contains 111 abused certificates, which is insufficient for in-depth analysis. Therefore, we first leverage the CA-published revocation reason codes as ground-truth references, and propose a new fine-grained classification method. By combining numerous auxiliary datasets (e.g., threat intelligence and company information), this method could classify abused certificates into five types: *Invalid Signature*, *Untrusted Certificates*, *Steal Certificates*, *Steal Developer IDs* and *Fake Developer IDs*. We applied this method to the base dataset, and obtained the largest labeled code-signing abuse dataset to date, comprising 43,286 certificates and 3,216,113 signed malicious samples. Using this dataset, we conducted an in-depth measurement of the code-signing ecosystem.

Main findings. Our analysis confirms that code-signing abuse remains severe, and the abusing techniques are evolving. While the low-tech type of abuse *Invalid Signature* still prevails (accounted for 89.5%, dominant before 2017), abusers are increasingly adopting advanced methods like *Steal Certificates* and *Steal Developer IDs*. The samples of such abuse types increased fourfold in the past 7 years (2017-2023) compared to the preceding 7 years (2010-2016). Malicious samples leveraging these advanced techniques can bypass client-side verification, significantly escalating the threat. Such abuse is widespread, affecting certificates from 114 countries issued by 46 CAs. While previous work [26] identified Symantec as the most affected CA in 2017, we revealed shifts in the ecosystem, with CAs like Sectigo and COMODO now facing greater threats. We also assessed the compliance of the current code-signing certificate revocation. Encouragingly, CAs have improved transparency, with 45.72% of post-2020 revocations providing specific reasons, which are valuable for inferring the causes of abuse. However, overall revocation deployment remains inadequate, with only 17.56% of certificates signed malware in our dataset being revoked. In addition, we first introduce a new challenge faced by CAs in revoking abused certificates: the difficulty in maintaining revocation infrastructure after the invalidation of CRL and OCSP signing certificates, a situation we refer to as "Ghost Certificates". It hinders CAs from revoking abused certificates, even when abuse is detected.

We then analyzed the tactics used by code-signing abusers, and found 5 strategies in the certificate application and signature deployment process to reduce costs or evade detection. For example, abusers deliberately apply through CAs in countries with lax identity verification to evade detection. They also employed dual-signature (10.17%, signing the same sample with certificates of different cryptographic algorithms) to enhance compatibility. Particularly noteworthy is the certificate polymorphism strategy, in which the same entity obtains multiple certificates from the same or different CAs using the same (or slightly modified) identity. We identified 3,484 polymorphic clusters, confirmed their wide usage by abusers (26.82%), and uncovered new obfuscation techniques like visual confusion to evade CA identity checks. For the first time, we discovered 315 real cases of malware using polymorphism to evade revocation checks by well-known CAs like Sectigo (related certificates have been revoked upon our report).

Our study highlights the current flaws in code signing practices, especially on the CA side, such as lax certificate content restrictions, lenient identity verification and unproactive abuse governance. We also propose feasible mitigations to increase the ecosystem transparency and proactive abuse supervision. To date, we have reported issues to the top 20 CAs by abuse volume. Sectigo, GlobalSign, and Entrust have confirmed our report and executed revocations. We hope this work could raise awareness and call for more standardized regulations of code-signing. Besides, we released the labeled dataset of abused certificates to facilitate future research (see Appendix I).

Contributions. Our main contributions include:

- We developed a new fine-grained classification method for code-signing abuse and built the largest dataset labeled with abuse types to date.
- We provided an in-depth analysis of the code-signing abuse ecosystem, revealed the current security state and disclosed various tactics used by real-world abusers.
- We identified flaws in the current CA code-signing practices, and provided recommendations to increase the transparency of the system and the proactivity of abuse regulation.

II. Background and Related Work

A. Overview of code-signing

Code-signing process. Code signing is vital for combating malware on mainstream operating systems. To sign software, a developer first obtains a certificate from a (trusted) CA after authentication, then uses the associated private key to sign a hash (e.g., SHA-256) of the software. The code-signing certificate and signature are bundled with the software and delivered to clients. A detailed structure of code-signing signature is provided in Appendix A. During download and installation, the operating system or antivirus software would verify the certificate and signature. Certificate validation authenticates the developer's identity, while signature verification ensures the integrity of the software, collectively strengthening security. For instance, the Windows verifier using the WinTrust APIs [39] warns users about software with missing, invalid, or untrusted signatures, highlighting potential installation risks.

Code-signing abuse. Code-signing certificates from CAs are theoretically supposed to be issued only to verified developers, safeguarding their benign software with signatures. The standard organization CA/Browser Forum (CA/B) explicitly mandates that CAs must not issue code-signing certificates for the signing of suspect code [16].

However, research and security incidents [26], [34] indicate that attackers would exploit the code-signing process, applying signatures to malware to evade alerts from operating systems

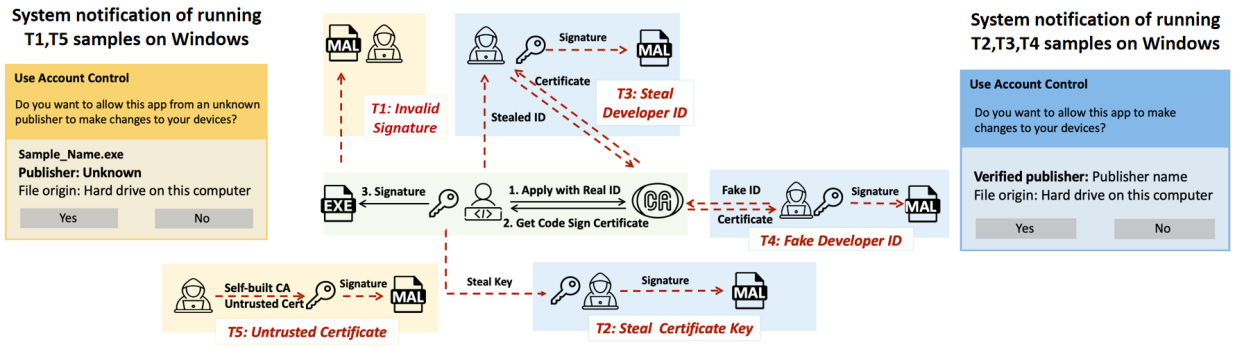

Fig. 1: Types of code-signing abuse.

and antivirus programs, to amplify the potency of their malware. *We refer to the situation where malicious software is equipped with code signatures as "code-signing abuse".*

To achieve their abuse objectives, attackers may employ a variety of methods. For instance, attackers have been known to use tools [44], [53], [54] to replicate TLS certificates from well-known websites or extract signatures from legitimate software, which are then embedded into malware. More sophisticated attacks exploit weaknesses in the code-signing certificate issuance and management process, e.g., stealing valid certificates (private keys) from legitimate companies to impersonate trusted applications [28], [34], [36], leveraging shell companies [23] or impersonating legitimate organizations to deceive CAs into issuing fraudulent certificates [22], [52]. We detail each abuse type and its impact in Section II-B.

Revocation mechanism. Code-signing PKI introduces a unique mechanism absent in Web PKI: the use of Time Stamp Authority (TSA) to record the exact creation time of the signature. This allows a signature to remain valid even after the certificate expires, as long as it was applied within the certificate's validity. Therefore, if code-signing abuse is not promptly governed, malware could continue to pass signature verification on client systems indefinitely.

Certificate revocation is an essential solution to counter code-signing abuse. Ideally, upon discovering a certificate used for signing suspicious code, the CA should conduct an investigation and publish revocation information once the abuse has been confirmed. Due to the TSA, the CA must specify the effective date of revocation, thus invalidating signatures made after that date. As with Web PKI, revocation information is published through Certificate Revocation Lists (CRLs) and the Online Certificate Status Protocol (OCSP). However, a previous study [27] has observed a significant delay in releasing certificate revocation information. There may be considerable numbers of certificates that have been used to sign malware and remain unrevoked (i.e., the abuse is still in effect).

B. Types of code-signing abuse

Abuse types. Code-signing abuse can enable cybercrimes like distributing malware, aiding attackers in the obfuscation of identity and evasion of scrutiny. "Obfuscation of identity" means that attackers use forged signatures and certificates that do not disclose their true identities. "Evasion of scrutiny" requires these signatures and certificates to be credible enough to pass signature validation checks. As this paper concentrates on code-signing abuse on the Windows platform, "pass signature validation checks" can specifically refer to the official Windows system verifier using the WinTrust APIs, and various antivirus engines on the Windows platform. Based on the techniques that attackers employ, we categorize code-signing abuse into five types (T1∼T5), as depicted in Figure 1.

Invalid Signature (T1) involves samples with unrecognized (incorrect format) or mismatched signatures. One common T1 strategy for an attacker without a valid code-signing certificate is to directly copy an existing signature and certificate from other software or web servers [44], [53], [54]. Typically, this method fails Windows' WinTrust API validation checks, resulting in an "Unknown developer" warning (as shown in Figure 1) to the user. Steal Certificate Key (T2) indicates that the abuser has obtained the legitimate software publisher's certificate private key or gained access to their signing machine, typically due to the publisher's inadequate protection of their private key. In T3 and T4, abusers obtain certificates from CAs by exploiting vulnerabilities in the identity validation process of CAs. Steal Developer ID (T3) involves obtaining certificates by appropriating the identities of other real entities or individuals. Fake Developer ID (T4) refers to creating fake identities, like shell companies that have not been legally registered. Both T3 and T4 aim to conceal the developers' true identities to prevent being traced and penalized once their signed malware is detected. Note that, since samples of T2∼T4 are all signed with code signing certificates issued by trusted CAs, they would trigger no anomaly warnings when executed on Windows clients. As depicted in Figure 1, users would be implied that the file was from "Verified Developer". Untrusted Certificate (T5) infers attackers signing samples with certificates from untrustworthy root certificates (self-built root certificates that are absent from the Trusted Root Program [6]). These samples would also trigger "Unknown Developer" warnings during execution.

Research scope. As mentioned above, we refer to the practice of attaching code signatures to malicious software as "code-signing abuse". As illustrated in Figure 1, T1 and T5 involve straightforward abuse methods that are easily detected by clients and trigger prominent user warnings, limiting their

effectiveness. In contrast, T2∼T4 represent scenarios where attackers have access to the private keys of certificates issued by public trusted CAs. Their validation results and user notifications on Windows are indistinguishable from benign software. Given their heightened stealth and broader impact, this work primarily focuses on analyzing the threat of code-signing abuse from T2∼T4. Moreover, attackers might employ more sophisticated techniques, such as certificate forgery via hash collisions exemplified by the 2012 Flame malware's MD5 collision [38]. However, such attacks have become largely impractical since Microsoft deprecated MD5 certificates in 2013. Accordingly, we exclude sophisticated attacks like hash collisions from our scope and leave them to future work.

C. Related work

Previous studies on Public Key Infrastructure security have primarily focused on Web PKI (HTTPS) due to its high data accessibility and mature regulations. Researchers can actively collect data by TLS scanning [13], [20], [21], [46] or passively analyze TLS traffic [1], [5], [45]. Public datasets like Censys [7] and Rapid7 [51] have also served studies on HTTPS deployment [15] and hijacking [2]. Certificate Transparency (CT) [17] initiative by Chrome also offers CA issuance records that facilitate public audits. In comparison, the code signing PKI ecosystem is more closed and opaque, posing constraints such as the difficulty in obtaining scalable datasets and ground-truth references, leading to the current insufficient research.

Among the few prior studies on code-signing PKI, Kotzias et al. [30] first analyzed Windows code certification abuse in 2015. At that time, code signing abuse was mainly associated with potentially unwanted program (PUP) rather than malware (only 5%-12% of abuse cases). In 2017, Kim et al. [26] conducted the first systematic study on code-signing abuse by malware. Their analysis was based on 111 abused certificates, which was primarily case-driven study rather than scalable measurements. At that time, a key challenge to do scalable analysis was the absence of ground-truth, and a follow-up study on the code-signing underground [31] noted that previous classification methods could be bypassed by attackers. In contrast, our work benefits from recent improvements in CA revocation transparency (as discussed in Section IV-C), allowing us to use CA-published revocation codes as the ground-truth. Based on this, we develop a scalable classification method, resulting in the largest and reliable dataset of fine-grained code-signing abuse to date. This allows us to analyze the ecosystem with in-depth measurements, especially for understanding the behaviors and strategies of abusers. Experience from other research areas (e.g., Telephony threats [18], [49], [59], DNS abuse [4], [19]) suggests that a deeper understanding of attacker behavior can help identify flaws and design effective defenses. Our fine-grained dataset enables us to achieve this for the first time in the code-signing field. In 2018, Kim et al. [27] evaluated the effectiveness of certificate revocation in code signing PKI. They still do not involve in-depth research on abuse types and behavioral analysis. Besides, the security industry has gained attention on code-signing abuses, but they predominantly published technical reports [56], [58] with superficial overview of the abuse, lacking in-depth analyses of abuse types, tactics, and other details.

III. Methodology

To investigate the current ecosystem of code-signing abuse, we first collected malicious software in the wild. As well-labeled datasets are scarce, we developed an approach leveraging signing-related features to label abused certificates into specific abuse types, and obtained so far the most extensive labeled code-signing abuse dataset. Figure 2 depicts the data processing workflow. This section details dataset sources, abuse type labeling, and certificate association.

A. Data collection

The primary data in this paper are malicious PEs with code signatures. There are two main sources used in prior studies: 1) datasets provided by commercial companies, such as Symantec [26], [30] and 2) publicly available datasets, e.g., VirusShare [30], [33]. Our datasets also originate from these two sources. We also collected complementary datasets and tools to extract code-signature features to identify abuse types.

> **Note:** In this paper, *malicious sample* denotes any sample that is labeled as either malware or a potentially unwanted program (PUP). *Malware* denotes clearly harmful software (e.g., trojans, ransomware, and backdoors). *PUPs* denotes software that is unwanted by users (e.g., adware and bundlers) and typically compromises privacy or weakens the computer's security, yet is generally regarded as less malicious than malware [29], [30].

Collect malicious samples. We collected all samples released by VirusShare between Oct. 2020 and Oct. 2024 (6,946,816 in total), a public malware repository also used in prior code signing studies [30], [33]. After filtering, we retained 176,968 signed PE samples. In addition, we collected 3,828,744 signed PE files (May 2006 - Sep. 2024) from a leading cybersecurity company. These samples originated from the company's client security software and vendor exchanges, and were flagged as malicious based on aggregated detections from multiple antivirus engines in the company's sample analysis system. Ultimately, the merged dataset comprises 3,962,788 signed PE files after 42,924 signed duplicates were removed based on sample hash. As previous studies used pre-2017 datasets [26], [30], our dataset aims to offer a more current view of the code-signing abuse landscape.

Extract code signing features. We developed a code signing extraction program based on pkcs7 [12], which supports dual signatures to retrieve the SignedData and X.509 certificates. Additionally, we extracted related information of samples from the Resource Section, such as ProductName and FileDescription. To further distinguish code-signing abuse types and gain insights into the abuse behaviors, we also extracted sample characteristics from the following aspects:

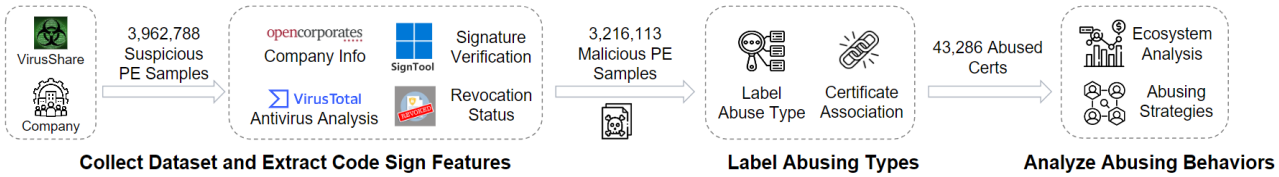

Fig. 2: Overview of the data processing flow.

TABLE I: Labeled dataset of abused samples and certificates.

Types of Abuse	Total		Malware		PUP	
	# Samples	# Certs	# Samples	# Certs	# Samples	# Certs
Invalid Signature (T1)	1,287,115	20,672	1,092,762	15,539	194,353	8,168
Certificates from Public CAs	1,913,973	23,252	228,206	11,797	1,685,767	15,647
Steal Certificate Key (T2)	21,991	284	7,010	238	14,981	97
Steal Developer ID (T3)	3,070	193	2,933	184	137	19
Fake Developer ID (T4)	1,480	125	1,071	114	409	18
Unspecified	1,887,730	22,650	217,469	11,261	1,670,261	15,513
Untrusted Certificate (T5)	15,035	8,259	9,872	7,989	5,163	340
Total	3,216,113	43,286	1,330,838	32,155	1,885,275	18,211

[1] For samples that use dual-signature, the two signatures may belong to different types of abuse. Thus these samples may be counted multiple times under different types of abuse.
[2] An abused certificate can be counted multiple times if it falls under different abuse types, such as T1 and T5, across various samples.

1) Antivirus scanning results. Although both sources label their samples as malicious, we further filtered the merged dataset to align criteria and reduce false positives. We scanned all collected samples using the antivirus engines on VirusTotal [57], and gathered information including the malware family, compilation time, and time of first observation. Following prior work [30], we retained only samples labeled as malicious by at least three antivirus engines (denoted as $c_{\text{mal}} \geq 3$), yielding 3,216,113 filtered malicious samples.

Considering that malware is more malicious and poses a greater threat than PUPs, we focus mainly on malware-related abuse. To this end, we adopted an approach similar to previous studies [30] to distinguish malware and PUPs, by constructing a list of 13 PUP-related keywords and determining the sample type based on their presence in the tags returned by 11 AV engines. A sample was classified as a PUP if at least half of the engines recognized it as such (denoted as $r_{\text{pup}} \geq 0.5$); the remaining samples are treated as malware. Ultimately, our dataset comprised 1,330,838 signed malware samples and 1,885,275 signed PUP samples. In Section V and Section IV, all measurement results, except for the certificate polymorphism analysis, are based on malware samples. The sensitivity analysis for c_{mal} and r_{pup} is presented in Appendix B.

2) Revocation information. When a certificate is abused to sign malware, the CA should promptly revoke it via CRL or OCSP, ideally specifying reasons like key compromise after thorough investigations. While CA revocations are often delayed [27], explicitly revoked certificates with detailed reason codes remain valuable for identifying abuse types.

Only revocation information published by trusted CAs is reliable. We extracted 23,252 entity certificates issued by trusted CAs from 1,913,973 samples (subset of 3,216,113) and obtained CRL links from their "Authorized Information Access" extension, resulting in 531 deduplicated CRLs. We attempted to access these CRLs through clients located in multiple countries (to circumvent potential inaccessibility due to region-specific blocking). Ultimately, CRL information was obtained for 97.27% of the 23,252 certificates, enabling verification of certificate revocation status. Discussion of reasons for CRL retrieval failures and their implications is provided in Appendix C. Totally, we identified 3,309 revoked certificates, of which 598 provided specific revocation reasons. These data will serve as ground truth to classify abuse behaviors.

3) Signature validation results. SignTool [40], based on the WinTrust APIs, is the official signing tool of Microsoft, offering functions such as code signing and signature verification. We employed SignTool to verify the signatures and certificate chains of collected samples, and recorded the returned resulting messages, as shown in Table V in Appendix D. Notably, some samples with recognizable signatures by our parsing program are not identified by the Windows SignTool due to non-standard signature format, returning a "No signature found" code. We regard them as the "low-tech" type of abuse ("Invalid Signature") and retain them for analysis.

4) Organization information. The legitimacy of the company registration of an applicant is crucial in differentiating between "Steal Developer ID" and "Fake Developer ID". OpenCorporates [48], the known largest open database of company information, offers in-depth details such as incorporation dates and operational status. OpenCorporates gathers its data primarily from official public sources, which ensures the reliability and provenance of the information. Although not exhaustive, this dataset aids in confirming the registration and existence of specific company entities or executives, thereby identifying fake developer identities. We parsed Subject CommonName field from certificates of the code signature samples

to query in that database and ended up with 167 registrations.

B. Label abuse types

After getting 3,216,113 malicious samples, we take the following steps to classify them into different abuse types.

Step-I: Identify *Invalid Signature* Samples. While the remained samples all contained code signatures as indicated by our parsing program, these signatures may not be recognized by SignTool (error message "No signature found" in Table V in Appendix D) or mismatched (error code 0x80096010 in Table V) with the code files. We categorized those samples as Invalid Signature (T1).

Step-II: Discover *Untrusted Certificate*. For the remaining samples with matched signatures, if their certificates were issued by untrusted CAs (error code 0x800B010A in Table V), they would be labeled as Untrusted Certificates (T5).

Step-III: Find *Steal Certificate Key*. All remaining samples are from attackers in possession of private keys of certificates issued by legitimate CAs. Then, to classify these into the T2∼T4 categories, we primarily rely on the revocation reasons stated by the CA in the revocation information. As per CA/B standards [16], CAs **MUST** promptly investigate and revoke any certificate found to be abused, recording the reasons in the revocation details. Of the 23,252 abused certificates from 1,913,973 samples remaining, we identified 3,309 revoked certificates, and 598 of these provide explicit revocation reasons. The term "KeyCompromise" indicates private key theft, and we identified 5,764 samples meeting the relevant criteria (T2).

Step-IV: Distinguish *Steal Developer ID* and *Fake Developer ID*. The revocation code "PrivilegeWithdrawn" signifies "directly obtained from CA". This code includes two abuse types - Steal Developer ID and Fake Developer ID, which we distinguish by verifying the existence of the holder's identity entity. We use the OpenCorporates database to confirm duly registered corporations. If the entity mentioned in the certificate is present in the database, we categorize the certificate as Steal Developer ID (T3). If the entity is not found, we deduce that the attacker has created a shell company identity (Fake Developer ID) to request a certificate from a CA (T4). In addition, certificates that do not receive a clear category label after the above categorization process will be marked as "Unspecified".

Ultimately, we obtained the labeled dataset as illustrated in Table I. It is the largest abused code-signing certificates dataset to date, with a scale exceeding that of any other known datasets [25] (111 certs).

C. Certificate association

Certificate polymorphism. Previous studies [31] have reported a phenomenon known as "certificate polymorphism" in code-signing abuse. This refers to the practice where the same entity applies multiple certificates from the same or different CAs with the same (or slightly modified) identity. By doing so, the abuser can obtain multiple certificates at a relatively low cost while evading scrutiny from CAs for revocation. This strategy saves on the financial costs of registering multiple companies and allows other certificates to remain active even if one is revoked, as CAs may not recognize them as coming from the same entity. We developed a correlation method based on the principle of certificate polymorphism, to trace various abused certificates back to the same certificate holder. This method not only facilitates subsequent analysis of abuse behavior, but also helps to label additional abused certificates.

Identify polymorphic certificates. The identification of the certificate holder relies on two key fields in the certificate content: the subject name and the public key. In other words, our association method operates on two assumptions: 1) certificates sharing the same public key originate from the same certificate holder; 2) certificates using the same or similar CommonName (CN) in the subject name come from the same certificate holder. Public key can be matched through direct field comparison. In contrast, the key challenge for CN-based association is to assess the similarity between CN fields. We first reviewed proposed methods in existing studies on this task, including: 1) measuring the normalized edit distance (NED) between CN fields [30], and 2) preprocessing (e.g., converting all letters to lowercase, removing non-alphabetic characters) for exact matching (PEM) [31]. These approaches all suffer from certain limitations: they cannot handle synonyms (e.g., Fuzhou Chuangyijiahe vs. Fujian Chuangyijiahe) or visually confusing letter substitutions (e.g., AppIe vs. Apple), which are commonly used evasion techniques in polymorphic certificates. Therefore, we leverage the LLM GPT-4 [47] to refine the outputs of these methods and improve recognition accuracy. Specifically, subject CN fields were extracted from certificates issued by public CAs and paired as inputs. Using the NED and PEM methods, we obtained 8,262 and 7,823 candidate CN pairs, respectively. The results of these two methods were then merged and fed into the GPT-4 LLM for refinement. We carefully designed prompts to guide its decisions based on prior research and our empirical observations (see Appendix E for details). In total, this process yielded 8,110 CN pairs that satisfied our matching criteria. After recognition, certificates with "similar" subject names would be grouped into one cluster. If two certificates from different clusters share a public key, we merge the clusters. We identified a total of 12,989 clusters, of which 3,484 exhibit polymorphism, indicating that the same holder has multiple different certificates. The polymorphism phenomenon will be explained in detail in Section VI.

We also evaluate the effectiveness of our LLM-based method and compare it against NED and PEM. As the evaluation dataset, we use all CN pairs that are labeled as "similar" by at least one of the three methods. First, we randomly sample 300 pairs for manual review and find that all annotations are consistent with the LLM's outputs. We then manually inspect only those pairs on which the three methods disagree. In this set, all correct identifications made by PEM are a subset of those made by our LLM-based method, while NED introduces 152 false positives. Finally, to address potential randomness in LLM outputs, we assess its stability and observe 99.66% agreement across three independently

repeated tests, as detailed in Appendix E. Overall, these results indicate that our approach to identifying similar common names outperforms existing methods.

Labeling Unspecified Certificates. Certificate association further refines our labeling in Section III-B. We observe that clusters obtained from certificate association contain abused certificates of only a single type, indicating that holders within a cluster typically employ one abuse method. For example, if two certificates are registered with the same fake identity and one is labeled Fake Developer ID, the other should be as well. Leveraging this insight, we start from certificates already labeled as T2∼T4 and propagate labels to unlabeled certificates sharing the same CN or public key. To be conservative, we apply this method only to certificates that have signed malware. Additionally, to evaluate this method, we conduct a hold-out experiment by dividing the labeled dataset into training and test subsets and propagating labels from the training set to the test set. Under a 1:9 train–test split, the method achieves 100% accuracy with 84.41% coverage, demonstrating the reliability of this method (see Appendix F for details).

Ultimately, this refinement of the labeling procedure yields 287 previously unspecified certificates (159 Steal Certificate Key, 65 Steal Developer ID, 63 Fake Developer ID), which are incorporated into the final results in Table I.

D. Limitations

Accuracy of revocation reasons. Empirical research on the specific reasons for certificate abuse is scarce. Therefore, the revocation reasons published by CAs, although not necessarily completely accurate, are the most credible references to identify abuse types. The CSBR [16] sets clear requirements for CAs' abuse investigations and certificate revocations, which supports the reliability of these records. Besides, RFC 5280 [8] specifies revocation-reason values but offers little guidance, so interpretations may vary among CAs. Our understanding aligns with the policies published by Microsoft [9] and Mozilla [42]. We also identified two CAs that publish detailed explanations, which are consistent with us.

Coverage of revocation reasons. Certificates that can be directly classified using CA-published revocation codes make up only approximately 20% of all revoked certificates, which is one limitation of our methodology. Nevertheless, the proportion of CAs providing specific revocation reasons has increased in recent years, improving our coverage for newer certificates: 31.90% of certificates issued after 2017 and 45.72% issued after 2020 include such reasons. In Section III-C, we further broaden the set of identifiable abuse types through certificate association. Overall, our labeled dataset of abused code-signing certificates is the largest to date.

Scope of abuse behaviors. Our identification of abused samples is based on VirusTotal detections. Consequently, highly sophisticated attacks that evade VirusTotal (e.g., certificate-collision attacks [38]) would not appear in our dataset, which constitutes a limitation of this study. Nevertheless, because our labeling relies on the aggregated decisions of multiple VirusTotal engines, we expect attacks that simultaneously evade all of them to be relatively rare.

Coverage of OpenCorporates. We use the OpenCorporates database to distinguish between T3 and T4. Although its coverage limitations might bias classification accuracy by overstate shell-company prevalence, OpenCorporates remains the most comprehensive dataset and is used in prior research [26]. We manually verified its coverage by using search engines to locate official government records for corporate entities. Evaluating 50 randomly selected entities from our dataset, we found that OpenCorporates could achieve a coverage of 90%.

Accuracy of certificate association. We leveraged the LLM-based method for associating and clustering abused certificates. A limitation of it is LLM's poor interpretability. However, in Section III.C, we evaluated the effectiveness of this method through manual inspection and a hold-out experiment, showing that it is currently the best available approach.

IV. Current Status of Code-signing Abuse

Using the above methodology, we collected 1,330,838 malware samples with code signatures, of which 1,113,634 (83.68%) labeled specific abuse types (T1∼T5), forming to date the largest labeled code-signing abuse dataset. Based on signing timestamps (see Appendix G for details), the majority of our samples (78.25%) were issued after 2017. Thus, the dataset enables an in-depth analysis of the current code-signing abuse ecosystem. This section discusses the types, scale, trends of abuse and the deployment of CA revocation.

A. Abuse type changes

Our labeled dataset with specific abuse types enables a more granular analysis of the abusing trends. It was found that the dominant abuse type was invalid signature (accounted for 89.5%, dominant before 2017), while only a few samples were signed by untrusted certificates. We plotted the number of abused certificates for T2∼T4 of code-signing over the past approximately 20 years in Figure 3. As shown in the main graph, after 2012, possibly due to the deployment of Windows signature verification mechanisms and malware protection strategies, attackers began to use more costly abuse ways, i.e., abusing certificates from legitimate CAs. The samples of such abuse types increased fourfold in the past 7 years (2017-2023) compared to the preceding 7 years (2010-2016). We further scrutinized the data related to legitimate CA certificate abuse from 2012 to 2024, and calculated the change in the number of abused certificates of each type, which is shown in the upper-left sub-figure. We omitted the cases where the revocation reason remains unspecified. We found a noticeable increase in the number of certificates involved in Fake and Steal Developer ID. Although these abuse types have been reported [26], it is clear that they have not been effectively curbed. In other words, we find the technical level of code signing abusers has advanced, which also imposes higher demands on the efficiency of detecting abuse.

We further analyzed the stealth capabilities of samples signed by certificates from different abuse types. VirusTotal

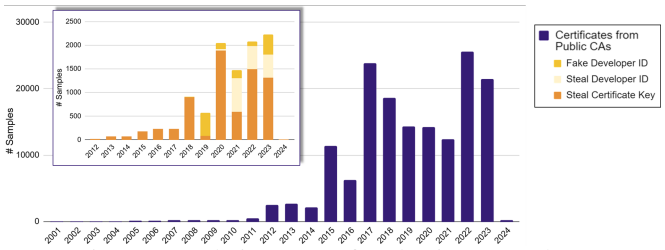

Fig. 3: Trends in types of code signature abuse.

uses multiple antivirus engines to detect the maliciousness of software. We discovered that the average number of engines that marked samples involving Steal Certificate Key, Steal Developer ID, and Fake Developer ID as malicious were 36.96, 41.42, and 34.44. It indicates that malware involving T2 and T4 can bypass certain antivirus engines. Additionally, we calculated the time interval from the signature date of the malware to the first detection by VirusTotal, which averaged 50.83 days. Specifically, the intervals for Steal Certificate Key, Steal Developer ID, and Fake Developer ID were 66.04 days, 25.48 days, and 39.47 days. This also suggests that samples involving stolen certificate keys have a longer active period in the wild and are more difficult to hunt by VirusTotal. Furthermore, we observed that 77.78% of certificates continue to sign benign samples after being stolen, indicating the abuse is covert for victims.

B. Scale and trends of abuse

Based on our dataset, we found that at least certificates from 114 countries issued by 46 CAs have been abused to sign malware samples, indicating a persistent problem in the current code-signing ecosystem. The next question is, from which CAs are the certificates being abused more severely?

To make an objective assessment, we need to consider the market share of the entire code-signing ecosystem, including both malware and benign samples. By collaborating with a leading cybersecurity company, we obtained a dataset of 15,190,141 benign software samples with signatures. It includes samples crawled from popular public software download sites and reported by their Windows client-side security software. The benign samples were signed from 2000 to 2023, with 88.23% after 2017, demonstrating their comparability to our malware sample dataset.

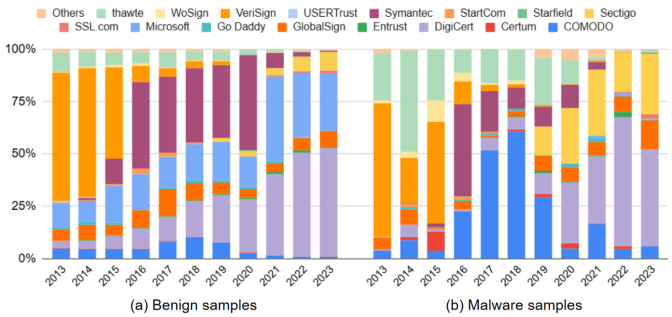

Fig. 4: CA distributions in signatures of benign samples and malware samples.

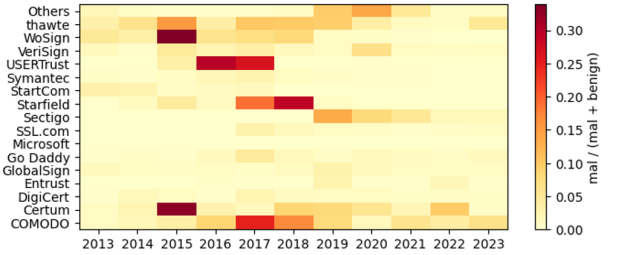

Fig. 5: Fraction of malware signatures in the overall sample by CA and year.

We extracted code-signing timestamps and the root CA information of the signing certificates from benign samples. To estimate the CA market share, we calculated the proportion of sample quantities and analyzed its changes over time, as shown in Figure 4(a). In the early stage, there were relatively few CA organizations providing code-signing services. Especially before 2015, Verisign held an absolute majority. Kim et al. noted that as of 2017, Symantec (including the Verisign and Thawte brands) dominated the code-signing ecosystem [26]. However, significant changes occurred in this ecosystem following Symantec's acquisition by DigiCert in 2020. In recent years, several new players have entered the code-signing ecosystem, particularly DigiCert, which has experienced rapid business growth and has now become the new mainstream.

We then compared the CA activeness in the code-signing abuse ecosystem. Since invalid signatures (T1) and untrusted certificates (T5) do not reflect the real situation of public CAs, we focus solely on certificates issued by public CAs. As shown in Figure 4(b), the abuse ecosystem is more diverse and less concentrated than benign signing, suggesting that the abuse behavior is prevalent across various CA organizations, even for non-popular ones. Besides, there are also a few CAs that have been more severely abused. We can see that, the proportion of abused certificates from Sectigo, COMODO, and WoSign significantly exceeds their benign market share. Figure 5 further shows, for each CA, the fraction of its issued certificates that were abused to sign malware. In addition to the three CAs discussed above, USERTrust, Starfield, and Certum also exhibit high abuse rates. However, the situation with Microsoft is the opposite, which may suggest that Microsoft has been highly effective in preventing abuse. We evaluated the prices of one-year Organization Validation (OV) certificates sold by the top 10 CAs by market share. Then the analysis shows no significant relationship between certificate price and either market share (p-value = 0.287) or the number of abused certificates (p-value = 0.262), with significance assessed at the conventional threshold of 0.05. Therefore, the high abuse rate may suggest potential security vulnerabilities that may be exploited by abusers.

C. Revocation situation

Revocation is expected to serve as a defense mechanism, distrusting abused certificates and thereby deactivating the malware they signed. Ideally, a certificate should be promptly reported to the CA once it has been found to be abused, and the

CA should investigate the abuse and issue the revocation notice with specific reasons upon confirmation. Prior studies [26], [27] have disclosed the insufficient and delayed deployment of revocation by code-signing authorities. We reassess the current revocation status with our dataset.

We found that CAs may have improved their regulation of certificate abuse in recent years but it remains inadequate. Based on our dataset, 11,797 certificates were abused to sign malware, 2,072 (17.56%) of which have been revoked, and 450 (21.72%) have been published with clear reasons for revocation. The revocation rate is lower compared to the results of Kim's work in 2017 [26] (27/111, 24.32%). Figure 6 shows the distribution of revocation statuses for the abused certificates. The horizontal axis represents its issuance dates, and the vertical axis indicates the number of malware samples each certificate signed. The left part shows non-revoked certificates and the right part shows revoked ones with colors indicating revocation reasons. It can be seen that certificates effectively revoked by CAs are concentrated among those issued in recent years (especially after 2019), indicating CAs' increasing focus on this issue. However, there has not yet been an effective revocation for older certificates, even those that are quite harmful, such as one (issued on Jul. 2, 2013) that signed 2,089 malware samples. Overall, out of 228,206 malware samples signed by certificates from public CAs, 178,376 samples had unrevoked certificates. As of this writing, 91,346 samples still could pass validation of SignTool (including revocation).

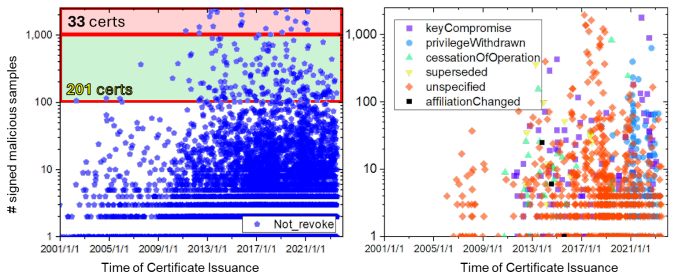

Fig. 6: Revocation status of abused certificates.

Besides, from the right part of Figure 6, we can see that CAs are providing more specific revocation reasons, allowing us to better label the dataset. Note that "cessationOfOperation" refers to revocation due to entity closure or service termination. "Superseded" means that a new, more appropriate certificate has replaced it. These revocation reasons do not match T2∼T4 discussed in this paper, so the corresponding certificates are also marked as "unspecified" in Table I.

Additionally, we found that many CAs still face challenges in setting an "effective revocation date", which determines the validity of binaries signed by a certificate. Binaries signed after this date are invalid, while those signed before remain valid. Hard revocation involves setting the date to the certificate's start of validity or earlier, invalidating all signed binaries. To address abuse, CAs must set the effective revocation date to that of the earliest instance of abuse or implement hard revocation.

To assess the effectiveness of revocation mechanisms, we checked abused samples that had timestamped signatures. Among 1,354 revoked certificates associated with these samples, 322 (23.78%) were set with incorrect revocation dates (i.e., not equal to or earlier than the earliest abuse time of the certificate), resulting in 2,792 (10.19%) abused samples remaining valid. The incorrect revocation rate is lower compared to the results of Kim's work in 2017 [26] (7/27, 25.93%). Besides, 941 certificates (69.50% of 1,354) employed a hard revocation mechanism. Although samples of "Steal Developer ID" and "Fake Developer ID" should all be subject to hard revocation, we found their hard revocation rates are 96.55% and 96.49% respectively.

D. Ghost Certificates

Section IV-C showed that the revocation rate of abused certificates remains unacceptably low, a shortcoming attributed in [27] to CAs' delayed identification of abuse and improper maintenance of revocation information. However, we observed an underappreciated but systemic cause rooted in the current timestamping mechanism and revocation infrastructure.

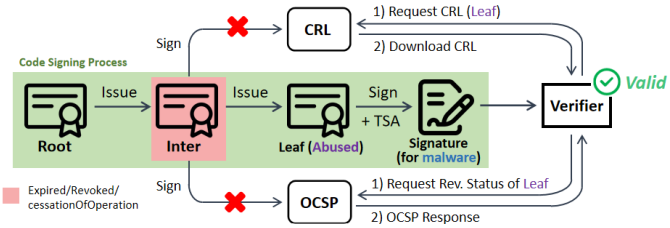

Fig. 7: Ghost Certificate.

We refer to this issue as "ghost certificates", i.e., code-signing certificates that have been abused but cannot be revoked, as shown in Figure 7. This problem arises from a structural limitation in the current revocation system. Verifying the revocation status of abused leaf certificates requires requesting CRL or OCSP. Both CRL and OCSP responses must be digitally signed to ensure authenticity. Although standards like RFC 5280 [8] and the CA/B Forum Baseline Requirements [16] do not strictly mandate that these responses be signed with the original issuer certificate, Microsoft clients default to using the intermediate certificate for validation as the issuer of CRL and OCSP response signatures. This raises a critical question: if the intermediate certificate has expired or been revoked, can it still be used to issue valid revocation responses for its previously issued certificates?

Notably, the certificate chain may remain valid despite the expiration or revocation of the intermediate certificate, due to the TSA mechanism and the fact that the certificate was issued before expiration/revocation. As a result, revoking the abused end-entity certificate remains essential, yet such revocation encounters structural obstacles. When we reported this issue to a major Certificate Authority, they confirmed that the expired intermediate certificate prevented them from

revoking the abused certificate we identified. Although the CA did not disclose the specific reason, its Certificate Policy (CP) indicates compliance with ETSI EN 319 411-1 [14], under which private keys of expired intermediate certificates shall be destroyed, making revocation impossible.

In our dataset comprising 9,725 certificates confirmed as abused yet remaining unrevoked, we assessed the prevalence of ghost-certificate conditions through analysis of CRL metadata. We ultimately identified 3,789 (38.96%) certificates meeting the conditions for ghost certificates, issued under 48 distinct issuer certificates. Among these certificates, 3,545 had issuer certificates that had already expired by the time of writing, 35 had issuers that were formally revoked (e.g., due to cessationOfOperation), and 411 were issued by CAs that had ceased operation entirely (e.g., StartCom), resulting in a complete cessation of revocation information publication. These empirical findings demonstrate that ghost certificates are not isolated anomalies but rather a prevalent and foreseeable outcome arising from the structural characteristics of the code-signing PKI, rather than from CA negligence.

Worse still, the number of ghost certificates will inevitably grow over time. Because every issuer certificate eventually expires, abused code-signing certificates that remain undetected or unrevoked at that point become effectively immortal: their associated malware will remain valid indefinitely. This structural limitation helps explain why revocation rates stagnate despite CA efforts. The challenges posed by ghost certificates are discussed in Section VII-A, with possible mitigation strategies in Section VII-B.

V. Abuse Strategies

The above analysis indicates that code-signing abusers remain active. To better identify the vulnerabilities and propose effective mitigation measures, we need an in-depth analysis of attackers' behaviors and strategies. This section describes the strategies we observed at multiple stages from our labeled dataset, including certificate application, signing, and dissemination of malware samples, aimed at evading checks, reducing costs, and expanding the impact of attacks.

A. Application strategy

Strategy-I: exploit differences in authentication policy leniency across countries. The "Country" field in the certificate subject name can help identify the geographical location of the certificate holder. We separately tallied the geographic distribution of certificates in benign and malware samples. In benign samples, certificate holders are distributed across 123 countries/regions, while in malware samples, they span 114 countries/regions. According to the number of certificates, the top three countries for benign holders are the United States (32.78%), China (18.13%), and Germany (8.11%), reflecting the prosperous software development industries in these three countries. However, by analyzing the geographical locations of malware developers, we found that there is a certain tendency in their distribution across countries.

The top three countries are China (CN), Korea (KR), and the United Kingdom (GB) for certificates in "Steal Certificate Key", the United Kingdom (GB), Slovenia (SI) and Denmark (DK) for "Steal Developer ID", and Russia (RU), Armenia (AM), Vietnam (VN) for "Fake Developer ID". We noticed that Armenia, Vietnam and Slovenia are only ranked 85th, 48th and 35th respectively in benign samples. This significant ranking discrepancy may indicate vulnerabilities in CAs' identity verification. After communicating with well-known CAs and looking up their policies, we learn that they often verify an entity's operational and physical existence through third-party sources, including local government public data and global public corporate databases. However, these data's credibility varies from country to country, resulting in data from less trustworthy countries affecting the global software ecosystem.

Strategy-II: use short-term certificates to reduce costs. CA providers typically offer certificates with varying validity periods, from one to three years, with longer durations commanding higher prices. For instance, the price for DigiCert's OV certificate is $539 for a one-year term, $1,024 for two years, and $1,536 for three years. We found that among the certificates used to sign benign samples, one-year certificates (350-380 days) accounted for 29.25%. In contrast, for abused certificates obtained directly from CAs (T3, T4), one-year certificates accounted for 84.22%, significantly higher than the normal proportion. Since these certificates are specifically applied for malware, they are usually put to use immediately after issuance.

Moreover, abusers commonly opt for short-term certificates to minimize costs and facilitate renewal, as the certificate may be revoked once its signed malware is detected. The phenomenon of certificate polymorphism, which will be discussed next, further illustrates abusers' tendency to maintain multiple short-term certificates.

Strategy-III: utilize polymorphic certificates to circumvent blocks. Certificate polymorphism denotes the scenario wherein a single entity holds multiple distinct certificates. This includes multiple stolen certificates from the same victim as well as multiple certificates acquired directly from CAs by the same abuser. Although previous studies [30], [31] have proposed the concept of certificate polymorphism, they did not explore its strategies and inherent threats in-depth. In this work, we proposed a clustering method to find polymorphic certificates (in Section III-C). Our study scope on certificate polymorphism includes all the abused samples where the attacker owns the trusted certificates (i.e., T2$\sim$T4), including 23,252 abused certificates and 1,913,973 samples of both malware and PUPs.

We find the phenomenon is widespread among abuse certificates. 13,747 out of 23,252 abused certificates (59.12%) exhibit polymorphism, forming 3,484 polymorphism clusters. Of these, 1,255 clusters contain certificates from multiple CAs and 68 clusters contain certificates from multiple countries/regions. Table II shows the top 5 clusters in all samples and malware samples (ordered by the number of certificates). The top cluster in all samples is typically PUP-dominated, but it invariably includes some malware. It can be seen

TABLE II: Top 5 clusters of certificate polymorphism in all samples/malware samples.

	Group	CN Example	#Cert	#CA	#CN	#Country	#PK	Revocation Rate	Issuance Time	EV Rate	#Sample	Submission Time	Samples Type
All	#15	Fried Cookie	784	25	337	26	530	0.64%	2003∼2022	1.02%	30,690	2009∼2024	Mainly PUP
	#108	Alpha Criteria	140	4	139	1	140	0%	2015∼2018	0%	4,116	2015∼2023	Mainly PUP
	#95	New Media Holdings	125	3	85	1	125	0%	2014∼2016	0%	7,811	2014∼2023	PUP
	#2825	Yu Bao	113	1	1	1	113	0%	2015∼2017	0%	187	2015∼2020	Mix
	#43	Superior Media	69	5	69	1	69	1.45%	2016∼2020	0%	2,683	2017∼2023	Mainly PUP
Malware	#3632	LADA	21	1	21	2	18	0%	2017∼2018	0%	30	2017∼2018	Mal
	#1252	AMCERT	18	1	3	1	18	55.56%	2021∼2023	0%	171	2021∼2024	Mal
	#2209	Sichuan Xunyou	18	6	1	1	18	61.11%	2013∼2019	16.67%	55	2017∼2020	Mal
	#1097	Financial Security	13	3	2	1	13	7.69%	2015∼2023	61.54%	2,496	2016∼2023	Mal
	#4494	5000 LIMITED	12	1	1	1	12	0%	2018	0%	32	2019∼2020	Mal

that, polymorphic clusters exhibit diversity in their forms. For example, the 1st cluster in all samples contains 784 abused certificates using 337 common names (CNs) in certificate subjects and 530 public keys (PKs). In comparison, the 4th cluster uses the same CN among 113 certificates with changing public keys. Top clusters dominated by malware are smaller but more recent, with some even using EV certificates to boost the credibility of malware. Notably, in most top malware clusters, the revocation status of abused certificates is poor, which may suggest that the polymorphic application has a certain evasive effect.

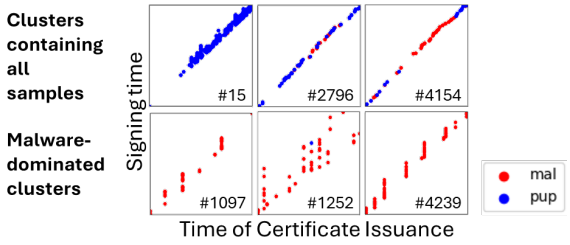

Fig. 8: Signing time distribution in top clusters.

Additionally, Figure 8 illustrates the relationship between certificate issuance and signing times in the top cluster. This shows that malware abusers typically apply for certificates at intervals, possibly to avoid suspicion from CAs. In contrast, PUP abusers continuously apply for and use new certificates, likely related to the customized demands of gray market clients. We provide a more in-depth introduction to polymorphic behavior in Section VI.

B. Implementation strategy

Strategy-I: leverage tactics as dual-signature to increase compatibility. Dual signing means applying two signatures to the same sample, typically to enhance its compatibility. For example, one signature uses SHA1 (compatible with older operating systems), and another uses SHA256. We discovered that dual signing is already used in code-signing abuse. Out of the 228,206 malware samples labeled as "Certificates Signed by Public CAs", 23,200 (10.17%) used dual-signing. This phenomenon indicates that code-signing abusers are becoming more diligent in increasing the usability of their samples. They not only strive to provide valid signatures but also use dual signing technology to enhance compatibility, further expanding the applicability and threat impact of their samples.
Strategy-II: sign malicious samples with multiple (evolving) abuse strategies. The Authenticode of a sample is a signature of its core code implementation and can be considered a unique identifier of one software. We found that abusers may use multiple code-signing abuse methods for the same software (with unique Authenticode). We found 9 Authenticodes involved in multiple abuse strategies, and 5 of them have been signed by both T1 and T2∼T4. We further compared the detection time by VirusTotal for the same Authenticode under different abuse types. The results showed that, five samples initially employed the basic tactic of forging signatures, and later transitioned to more sophisticated abuse types after being detected by VirusTotal. This suggests that abusers could progressively enhance their techniques to evade detection methods, which coincides with the findings of abusing type changes we discussed in Section IV-A.

VI. Certificate Polymorphism

In this section, we perform a detailed analysis of certificate polymorphism, one of the most unique and critical strategies adopted by code-signing abusers. We extend our study scope to include all T2∼T4 samples (including unspecified samples), covering both malware and PUPs. The inclusion of PUPs is due to two reasons: firstly, this phenomenon was initially observed in PUPs, with fewer instances in malware at the time; secondly, we found that malware and PUPs are often mixed in large clusters. We discuss how abusers apply for polymorphism certificates, and real-world cases where polymorphism facilitates the circumvention of CA checks in certificate application and revocation.

A. Polymorphism patterns

Certificate polymorphism consists of two patterns: "Public Key Reuse", where multiple certificates share a pair of public and private keys, and "Subject Obfuscation", where certificates have similar subject names. Both patterns indicate that the holders of these certificates point to the same entity. Below we discuss the details of these patterns based on our observation.
Public key reuse. We found 745 certificates (3.20%) exhibit public key reuse, involving 226 distinct public keys. The most common instance (129 public keys) of reuse is found in dual-signing certificates, where the two certificates use the same key but different signing algorithms. Another common scenario (56 public keys) of key reuse occurs during certificate renewal, where holders may reuse the key from the old certificate when applying for the new one.

In addition to the above common scenarios, we also discovered abnormal cases of public key reuse. A particularly notable

TABLE III: Obfuscation strategies in certificate polymorphism.

Strategy	Ratio	Example
Abbreviation Replacement	35%	Monitor, OOO Monitor, LLC
Case Substitution	35%	HASTINGS INTERNATIONAL B.V. Hastings International B.V.
Punctuation Change	16%	Onekit Internet S,L Onekit Internet S.L
Word Segmentation	5%	Suzhou MorningSun IT LLC Suzhou Morning Sun IT LLC
Visual Confusion	5%	STELLAR PC SOLUTIONS STELLAR PC SOLUTIONS

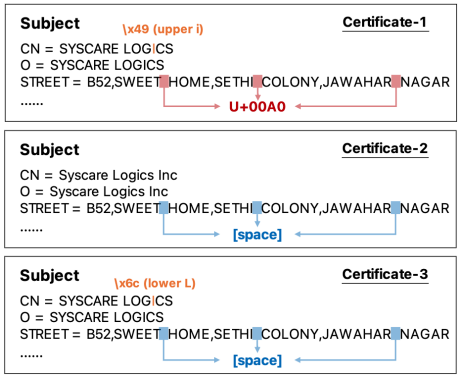

Fig. 9: Polymorphic certificates with special characters.

case is cluster #15 in Table II, which includes 251 certificates with the same public key, almost issued by Thawte. Despite these certificates using 89 subject names and originating from 26 different countries, their signed samples all belong to the same family, InstallCore. InstallCore is an adware platform from Israel often bundled with software installations, known for delivering PUPs and even malware. It offers certificate application and code-signing services for PUP developers. We hypothesize that, for operational convenience, InstallCore uses the same public key across different applicants. This practice contravenes the code-signing certificate issuance principle, as it involves issuing certificates to intermediaries rather than the actual entities, indicating potential authentication issues.

Subject obfuscation. We also observed that the abuser may apply for multiple certificates using "similar" common names by carefully modifying subject fields, which can be considered as a form of obfuscation. This practice was quite prevalent, with 5,999 certificates (25.80%) exhibiting this behavior. The top 5 frequently observed obfuscation strategies are summarized in Table III. The most common strategy is altering the abbreviations (35%) of company names. For instance, "Limited Liability Company" is abbreviated as LLC in English, but it is OOO in Russian. Notably, visual confusion is a more clear malicious category, e.g., replacing the "I" in "SOLUTION" with a lowercase "l", which is a common phishing tactic. Besides, we identified cases of concealed obfuscation using special (e.g., unprintable) characters. As shown in Figure 9, the applicant utilized visual confusion ("i" and "l") to construct cert-3 based on cert-2, and also created a more concealed cert-1 by replacing the original space characters with "U+00A0", which is an unprintable character and could be rendered the same as a space. Therefore, cert-1 visually seems indistinguishable from cert-2, although they are different at the encoding level. Similarly, "U+00AD" is also an unprintable character and could be rendered the same as "-". We identified "U+00A0" in 20 certificates' subject name and "U+00AD" in 3 certificates.

Subject obfuscation assists abusers in bypassing identity verification and certificate revocation. Abusers may employ this method for two reasons: 1) creating shell companies or masquerading as legitimate companies is costly, so abusers seek to obtain as many certificates as possible under the same identity; 2) when abuse is detected, it is difficult to readily identify other abused certificates linked to that entity based solely on the certificate information.

B. Case study

This case study shows the use of certificate polymorphism to evade revocation mechanisms. Kotzias et al. [30] demonstrated how PUP clusters exploit this technique. As abuse methods evolve, malware has begun to use the same approach. For the first time, we identified real-world instances of malware leveraging polymorphism to bypass revocation checks.

Inconsistent revocation. We attempted to identify instances of certificate polymorphism circumventing CA verification in clustering results, and identified 315 (9.04%) clusters showing inconsistent revocation, meaning some certificates used for signing malware were revoked while others remained active. Below, we will introduce the details by case studies.

According to CA/B Forum requirements [16], CAs are required to track entities signing suspect code and deny them new certificates (see **Req.1** in Appendix H). However, current enforcement is insufficient, allowing abusers to exploit certificate polymorphism to continue application. One case is cluster #1252 (in Table II) with 18 abused certificates. The earliest malware sample signed with these certificates was detected by antivirus engines on Jun. 25, 2021. Sectigo subsequently performed a hard revocation of the two earliest certificates. The attacker then continuously obtained 16 new certificates from Sectigo under fake identities. Only 7 were eventually revoked, highlighting Sectigo's failure to enforce the suspect entity list during issuance and revocation.

High-risk case. We also found polymorphic certificates facilitating high-risk APT campaigns. Cluster #1051 comprises 11 unrevoked certificates from 4 CAs with consistent subject names, responsible for 93 malware and 25 PUP samples. Notably, one certificate signed a sample linked to APT27 [35]. Issued in 2016, this certificate initially signed more than 60 benign samples before pivoting to malware, eventually signing the APT sample shortly before expiration. We confirmed with its issuing CA that this is a "ghost certificate", which makes revocation impossible. Although Windows blocks this specific APT sample, our tests confirm that other malware signed by this certificate still passes Windows' signature verification.

VII. Discussion

A. Root cause and weaknesses

The security of code-signing relies on multiple factors. Prior research in 2017 [26] proposed three main deficiencies: inadequate client-side protections, publisher-side key mismanagement, and CA-side verification failures. So far, notable enhancements have been made on the client and publisher sides, such as Windows SmartScreen [41] and the industry standard for storing keys on certified hardware modules [11]. But CAs still fall short in defending against code-signing abuse. Specifically, our findings reveal following weaknesses:

1) Lack of strictness and standardization in issuance. We found that CAs' verification of applicants' identities during the issuance process may have lacked scrutiny, enabling cases of Steal Publisher ID and Fake Publisher ID. The verification may rely on information from third-party databases, and flaws in these databases pose security risks to the issuance process. Besides, we found cases where intermediaries applied for and managed certificates on behalf of developers. Intermediaries might reuse keys across identities, posing security risks to key management. In certificate polymorphism, we found that abusers often make minor adjustments to subject names during the application to obtain multiple certificates under one identity. One common method is to insert invisible or non-printable characters, which can be prevented if CAs standardize the permitted range of characters in the fields during the issuance.

2) Unproactive abuse governance. We found CAs currently still adopt a "passive" role in governing code-signing abuse. Although CA/B has explicitly mandated that entities known to have intentionally signed suspect code should not be issued new certificates [16], there seems to be a lack of proactive identification of high-risk entities. In certificate polymorphism, we discovered that the same abuser, after having a certificate revoked, can continue to apply for new certificates by slightly altering the subject name. CAs also primarily rely on reports from subscribers and third parties to initiate abuse investigations, rarely taking proactive measures to monitor issued certificates. Victims may also be unaware of the abuse (e.g., private key leakage), leading to delayed revocation.

Besides, CAs may also be affected by design flaws within the code-signing PKI: 3,789 (38.96%) certificates satisfy the ghost-certificate conditions, whose invalid issuer certificates hinder revocation even when abuse is detected, thereby hindering effective mitigation. The root cause of this issue stems from two factors. First, the current standards and best practices that CAs follow impose weak requirements on revocation processes and fail to account for the difficulty of maintaining revocation services once issuer certificates become invalid. Second, Windows clients rely solely on certificate-specified CRL/OCSP, without any additional revocation mechanisms, leaving ghost certificates effectively non-revocable.

B. Mitigation suggestions

We outline practical measures for CAs, OS vendors, and security systems to address the identified issues.

For CAs. CAs should take concrete actions across issuance, revocation, and abuse governance. 1) *Increase transparency of revocation and issuance.* Increasing transparency will facilitate a better code-signing audit. For revocation, we recommend that CAs disclose not only the reason codes but also detailed information of relevant entities. This would help align with the policy of not issuing certificates to high-risk entities [16]. For the issuance, using artifact and identity logs under frameworks like Sigstore [43] can support audits and help legitimate certificate holders detect abuse. 2) *Proactively detect abusive behavior.* We expect CAs to take a more proactive role in abuse governance by considering the issuance history of entities, proactively using antivirus engines to monitor malicious signing activities, and auditing polymorphic certificates upon discovering abuse and timely revoking abused certificates. 3) *Establish standards for certificate subject names.* We recommend that the Common Name field of the certificate subject should only contain meaningful strings that can indicate the applicant identity. The fields should reject visually ambiguous letters, meaningless punctuation, or non-printable characters, to mitigate the impact of polymorphic abuses.

For operating system vendors. The operating system's signature verification logic may need to be adjusted, especially for *mitigating "ghost certificates"*. Beyond updating standards to instruct CAs not to use intermediate certificates in code-signing chains to sign CRL/OCSP responses, Windows should adjust its signature validation logic by decoupling CRL/OCSP checks from the code-signing chain, thereby enabling CAs to update revocation infrastructure or rotate keys independently.

For security systems. Client side may need to rely on security systems to *adopt proactive abuse governance*. As operating system checks may be insufficient to counter abuse, security systems can play a more proactive role. For example, by aggregating threat intelligence to maintain blocklists of high-risk certificates and assist users in blocking related malware. SmartScreen [41] is one such mechanism: it evaluates certificate reputation to block potential abuse.

C. Discussions with certificate authorities

We have emailed all reachable CAs except those that were unreachable due to mergers, acquisitions or closures. All top 20 CAs with the most abused certificates have been notified. So far, Sectigo, GlobalSign and Entrust have confirmed the reported certificates were indeed at risk of abuse and have revoked them. They also acknowledge the need for more proactive abuse mitigation. Some CAs have tried flagging suspicious identities based on abuse patterns but with limited success. Our clustering approach may help identify high-risk subscribers. During our communication, we confirmed the existence of ghost certificates. A leading CA confirmed that some of the abused certificates we reported were issued by a certificate that is expired, which makes revocation impossible.

VIII. Conclusion

Code signing is a crucial mechanism for verifying the publisher's identity and ensuring the code's integrity. However,

code-signing abuse poses a significant threat. This paper conducted a large-scale measurement study of the real-world ecosystem of code-signing abuse. We developed a fine-grained classification method for abused certificate types, obtaining the largest labeled dataset of abused certificates (43,286 certs) to date. Utilizing this dataset, we conducted an in-depth analysis of the code-signing ecosystem and abusers' behaviors, revealing a series of abuse strategies. We thoroughly analyzed certificate polymorphism behavior and identified real cases to evade detection. Finally, we offer targeted recommendations to mitigate the abuse of code signing.

Acknowledgment

We thank our shepherd and all reviewers for their valuable suggestions to improve this paper. We are also grateful to our industry partner for supporting this work with the dataset. This work is supported by the National Natural Science Foundation of China (62302258) and Zhongguancun Laboratory. Haixin Duan is supported by the Taishan Scholarship Program. Yiming Zhang and Lingyun Ying are both the corresponding authors.

Ethics Considerations

Three points of our experiments may raise ethical concerns: 1) the collection and analysis of malicious datasets, 2) exposing weaknesses in current code signing practices and 3) releasing labeled dataset. We refer to authoritative ethical guidelines such as the Menlo Report [24] and carefully design our experiments to minimize potential risks.

First, we followed the recommendations outlined in [3], and used public datasets and company datasets to collect malicious samples. As for data processing from public datasets, all downloaded data related to the malicious samples was securely stored on a dedicated experimental server to prevent leaks. After extracting relevant features, the original executables of the malicious samples were deleted. The company's data comes from a security company with which we have a research partnership. The researchers are interns who complete the relevant data analysis within the company. Notably, all processing and analysis tasks were conducted on a researcher-controlled server. Second, for the vulnerability disclosure, we have emailed all reachable CAs, including the top 20 with the most abused certificates. We have actively engaged with them to share the problematic signing certificates and revocation information we have identified. Recognizing the complexity of CA ownership structures(e.g., acquisition chains), we used CCADB records to identify each CA's final owner. For example, the final owner of Comodo, USERTrust, Intel External Issuing CA 7B, and OneSign OV Code Signing CA is Sectigo. We also encountered more complex cases—for instance, GoGetSSL certificates involve both Sectigo and DigiCert—and GoGetSSL confirmed that our abuse reports were forwarded to the appropriate owners. So far, we have received responses from three mainstream CAs (see Section VII-C for details). Last, we released the labeled abused certificate dataset. To mitigate ethical risks, we only disclosed the certificates that have signed malware.

References

[1] Mustafa Emre Acer, Emily Stark, Adrienne Porter Felt, Sascha Fahl, Radhika Bhargava, Bhanu Dev, Matt Braithwaite, Ryan Sleevi, and Parisa Tabriz. Where the wild warnings are: Root causes of chrome https certificate errors. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1407–1420, 2017.

[2] Gautam Akiwate, Raffaele Sommese, Mattijs Jonker, Zakir Durumeric, kc claffy, Geoffrey M. Voelker, and Stefan Savage. Retroactive identification of targeted DNS infrastructure hijacking. In Chadi Barakat, Cristel Pelsser, Theophilus A. Benson, and David R. Choffnes, editors, *Proceedings of the 22nd ACM Internet Measurement Conference, IMC 2022, Nice, France, October 25-27, 2022*, pages 14–32. ACM, 2022.

[3] Mark Allman and Vern Paxson. Issues and etiquette concerning use of shared measurement data. In Constantine Dovrolis and Matthew Roughan, editors, *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007*, pages 135–140. ACM, 2007.

[4] Sumayah A. Alrwais, Kan Yuan, Eihal Alowaisheq, Zhou Li, and XiaoFeng Wang. Understanding the dark side of domain parking. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 207–222. USENIX Association, 2014.

[5] Bernhard Amann, Matthias Vallentin, Seth Hall, and Robin Sommer. Extracting certificates from live traffic: A near real-time ssl notary service. Technical report, Citeseer, 2012.

[6] CCADB. Microsoft included ca certificate list. https://ccadb.my.salesforce-sites.com/microsoft/IncludedCACertificateReportForMSFT, 2024. Accessed: 2025-11-27.

[7] Censys. The censys bigquery dataset. https://support.censys.io/hc/en-us/articles/360038759991-BigQuery-Introduction, 2023.

[8] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and W. Timothy Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. *RFC*, 5280:1–151, 2008.

[9] Microsoft Corporation. Troubleshooting certificate status and revocation. https://learn.microsoft.com/en-us/previous-versions/tn-archive/cc700843(v=technet.10)?redirectedfrom=MSDN, 2009. Accessed: 2025-11-27.

[10] X de Carné de Carnavalet and Mohammad Mannan. Killed by proxy: Analyzing client-end TLS interception software. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[11] Digicert. New private key storage requirement for code signing certificates. https://knowledge.digicert.com/general-information/new-private-key-storage-requirement-for-standard-code-signing-certificates-november-2022, 2022. Accessed: 2025-11-27.

[12] Doowon. PKCS7 signedData parser (for authenticode). https://github.com/doowon/pkcs7. Accessed: 2025-11-27.

[13] Zakir Durumeric, James Kasten, Michael D. Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In Konstantina Papagiannaki, P. Krishna Gummadi, and Craig Partridge, editors, *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pages 291–304. ACM, 2013.

[14] EN ETSI. 319 411-1 v1.5.1 (2025-04) electronic signature and infrastructures (esi). *Policy and security requirements for Trust Service Providers issuing Certificates*, 2025.

[15] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring https adoption on the web. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1323–1338, 2017.

[16] CA/BROWSER FORUM. Baseline requirements for the issuance and management of publicly trusted code signing certificates. https://cabforum.org/wp-content/uploads/Baseline-Requirements-for-the-Issuance-and-Management-of-Code-Signing.v3.7.pdf. Accessed: 2025-11-27.

[17] Google. Certificate transparency. https://certificate.transparency.dev/. Accessed: 2025-11-27.

[18] Payas Gupta, Bharat Srinivasan, Vijay Balasubramaniyan, and Mustaque Ahamad. Phoneypot: Data-driven understanding of telephony threats. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[19] Shuang Hao, Matthew Thomas, Vern Paxson, Nick Feamster, Christian Kreibich, Chris Grier, and Scott Hollenbeck. Understanding the domain registration behavior of spammers. In Konstantina Papagiannaki, P. Krishna Gummadi, and Craig Partridge, editors, *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pages 63–76. ACM, 2013.

[20] 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 Patrick Thiran and Walter Willinger, editors, *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11, Berlin, Germany, November 2-, 2011*, pages 427–444. ACM, 2011.

[21] Lin Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. Analyzing forged SSL certificates in the wild. In *2014 IEEE Symposium on Security and Privacy*, pages 83–97, 2014.

[22] Spence Hutchinson. Hunting certified imposters. https://i.blackhat.com/SecTor-2024/Sector-24-Hutchinson-Hunting-Certified-Imposters.pdf, 2024. Accessed: 2025-11-27.

[23] Security Intelligence. MegaCortex. https://securityintelligence.com/posts/from-mega-to-giga-cross-version-comparison-of-top-megacortex-modifications/. Accessed: 2025-8-1.

[24] Erin Kenneally and David Dittrich. The menlo report: Ethical principles guiding information and communication technology research. *Available at SSRN 2445102*, 2012.

[25] Doowon Kim, Bum Jun Kwon, and Tudor Dumitras. Signed malware - The dataset. https://users.umiacs.umd.edu/~tdumitra/signedmalware/ccs17/ccs17.html. Accessed: 2025-11-27.

[26] Doowon Kim, Bum Jun Kwon, and Tudor Dumitras. Certified malware: Measuring breaches of trust in the windows code-signing PKI. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1435–1448. ACM, 2017.

[27] 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 William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 851–868. USENIX Association, 2018.

[28] Hitomi Kimura, Ryan Soliven, Ricardo Valdez III, Nusrath Iqra, and Ryan Maglaque. RedLine/vidar abuses EV certificates, shifts to ransomwaree. https://www.trendmicro.com/en_us/research/23/i/redline-vidar-first-abuses-ev-certificates.html. Accessed: 2025-11-27.

[29] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring pup prevalence and pup distribution through pay-per-install services. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 739–756, 2016.

[30] 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, New York, NY, USA, 2015. Association for Computing Machinery.

[31] Kristián Kozák, Bum Jun Kwon, Doowon Kim, Christopher S. Gates, and Tudor Dumitras. Issued for abuse: Measuring the underground trade in code signing certificate. *ArXiv*, abs/1803.02931, 2018.

[32] 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, 2018.

[33] Bum Jun Kwon, Sanghyun Hong, Yuseok Jeon, and Doowon Kim. Certified malware in south korea: A localized study of breaches of trust in code-signing PKI ecosystem. In Debin Gao, Qi Li, Xiaohong Guan, and Xiaofeng Liao, editors, *Information and Communications Security - 23rd International Conference, ICICS 2021, Chongqing, China, November 19-21, 2021, Proceedings, Part I*, volume 12918 of *Lecture Notes in Computer Science*, pages 59–77. Springer, 2021.

[34] Malwarebytes. Stolen Nvidia certificates used to sign malware: here's what to do. https://www.malwarebytes.com/blog/news/2022/03/stolen-nvidia-certificates-used-to-sign-malware-heres-what-to-do, 2022. Accessed: 2025-11-27.

[35] Mandiant. Advanced Persistent Threats (APTs). https://www.mandiant.com/resources/insights/apt-groups. Accessed: 2025-11-27.

[36] Rahat Masood, Um-e-Ghazia, and Zahid Anwar. SWAM: stuxnet worm analysis in metasploit. In *2011 Frontiers of Information Technology, FIT 2011, Islamabad, Pakistan, December 19-21, 2011*, pages 142–147. IEEE Computer Society, 2011.

[37] Microsoft. Windows authenticode portable executable signature format. https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/authenticode_pe.docx, 2008. Accessed: 2025-11-27.

[38] Microsoft. Flame malware collision attack explained. https://msrc.microsoft.com/blog/2012/06/flame-malware-collision-attack-explained/, 2012. Accessed: 2025-11-27.

[39] Microsoft. WinVerifyTrust function (wintrust.h). https://learn.microsoft.com/en-us/windows/win32/api/wintrust/nf-wintrust-winverifytrust, 2021. Accessed: 2025-11-27.

[40] Microsoft. SignTool. https://learn.microsoft.com/en-us/windows/win32/seccrypto/signtool, 2024. Accessed: 2025-11-27.

[41] Microsoft. Microsoft Defender SmartScreen. https://learn.microsoft.com/en-us/windows/security/operating-system-security/virus-and-threat-protection/microsoft-defender-smartscreen/, 2025. Accessed: 2025-11-27.

[42] MozillaWiki. CA/Revocation reasons. https://wiki.mozilla.org/CA/Revocation_Reasons. Accessed: 2025-11-27.

[43] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. Sigstore: Software signing for everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2353–2367, 2022.

[44] Paranoid Ninja. CarbonCopy. https://github.com/paranoidninja/CarbonCopy. Accessed: 2025-11-27.

[45] Edward Oakes, Jeffery Kline, Aaron Cahn, Keith Funkhouser, and Paul Barford. A residential client-side perspective on ssl certificates. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 185–192. IEEE, 2019.

[46] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. TLS Proxies: Friend or Foe? In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, New York, NY, USA, 2016. Association for Computing Machinery.

[47] Openai. Openai GPT-4. https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo. Accessed: 2025-8-1.

[48] OpenCorporates. OpenCorporates. https://opencorporates.com/. Accessed: 2025-11-27.

[49] Sathvik Prasad, Elijah Robert Bouma-Sims, Athishay Kiran Mylappan, and Bradley Reaves. Who's calling? characterizing robocalls through audio and metadata analysis. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 397–414. USENIX Association, 2020.

[50] Ram Sundara Raman, Leonid Evdokimov, Eric Wurstrow, J. Alex Halderman, and Roya Ensafi. Investigating large scale HTTPS interception in Kazakhstan. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[51] Rapid7 Labs. Open Data. https://opendata.rapid7.com/, 2023.

[52] Reversinglabs. A new kind of certificate fraud: Executive impersonation. https://www.reversinglabs.com/blog/digital-certificates-impersonated-executives-as-certificate-identity-fronts, 2019. Accessed: 2025-11-27.

[53] Secretsquirrel. SigThief. https://github.com/secretsquirrel/SigThief. Accessed: 2025-11-27.

[54] Tylous. Limelighter. https://github.com/Tylous/Limelighter. Accessed: 2025-11-27.

[55] VirusShare. VirusShare. https://virusshare.com/. Accessed: 2025-11-27.

[56] VirusTotal. Deception at a scale. https://blog.virustotal.com/2022/08/deception-at-scale.html. Accessed: 2025-11-27.

[57] VirusTotal. VirusTotal. https://www.virustotal.com/. Accessed: 2025-11-27.

[58] Mike Wood. Want my autograph? The use and abuse of digital signatures by malware. In *Virus Bulletin Conference September*, pages 1–8, 2010.

[59] Yiming Zhang, Baojun Liu, Chaoyi Lu, Zhou Li, Haixin Duan, Shuang Hao, Mingxuan Liu, Ying Liu, Dong Wang, and Qiang Li. Lies in the air: Characterizing fake-base-station spam ecosystem in china. In Jay

Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 521–534. ACM, 2020.

APPENDIX

A. Overview of the Authenticode signature format

Authenticode signatures can be embedded in a Windows PE file at a specific location defined by the Certificate Table entry in the Optional Header Data Directories [37]. During the signing process, the algorithm used to compute the Authenticode hash for the file intentionally excludes certain PE fields, ensuring that modifications to these fields do not alter the hash value. Figure 10 offers a simplified illustration of how the Authenticode signature is embedded within a Windows PE file. It highlights the location of the embedded signature and identifies the PE fields excluded from the hash computation. Additionally, it indicates the key fields that were primarily considered when collecting the signature dataset for this study.

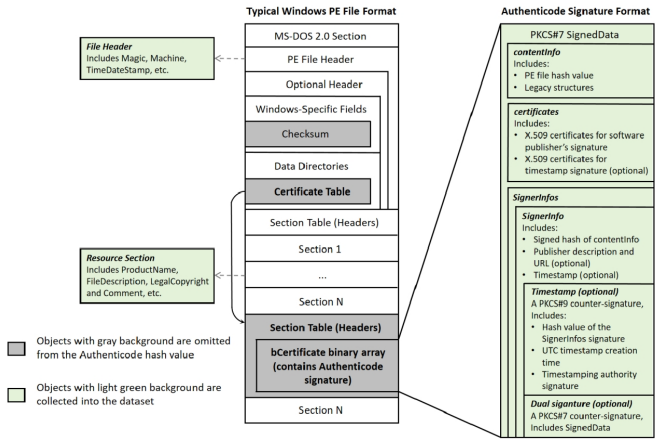

Fig. 10: Overview of the Windows PE file format and the Authenticode signature format.

B. Sensitivity analysis of data processing

We conducted a sensitivity analysis of the thresholds (c_{mal} and r_{pup}) used in our data processing pipeline to assess their impact on the downstream abuse classification. Although the choices of these two thresholds followed prior work [30], we aimed to examine how varying them affects the labeled sample dataset constructed in this paper.

Filter malicious samples. c_{mal} denotes the number of antivirus engines that label a sample as malicious. When selecting malicious samples, we set the threshold $c^*_{\text{mal}} = 3$ and retain only samples with $c_{\text{mal}} \geq c^*_{\text{mal}}$. This yields a labeled dataset with 3,216,113 malicious samples and 43,286 abused certificates. We then evaluate how the labeled dataset changes when the threshold is varied to $c^*_{\text{mal}} \pm 1$, as summarized in Table IV. Overall, changing the threshold to $c^*_{\text{mal}} = 2$ or $c^*_{\text{mal}} = 4$ has only a marginal effect on the labeled dataset: the total number of malicious samples varies by $+0.43\%$ and -0.22%, respectively, relative to the baseline setting.

Distinguish malware and PUPs. r_{pup} denotes the fraction of antivirus engines that label a sample as PUP. When distinguishing malware and PUPs, we set the threshold $r^*_{\text{pup}} = 0.5$ and classify a sample as PUP only if $r_{\text{pup}} \geq r^*_{\text{pup}}$. This results in 1,330,838 malware samples and 1,885,275 PUP samples. We further examine how the labeled dataset is affected when the threshold is adjusted to $r^*_{\text{pup}} \pm 0.1$, as reported in Table IV. In particular, setting r^*_{pup} to 0.4 or 0.6 only slightly perturbs the labeled dataset: certificate entries change by at most 4.35%, with an average change of 2.25%. Our experiments indicate that the threshold adopted in data processing does not materially affect the subsequent data analysis.

C. Details on fetching CRLs

We began the retrieval process with 531 deduplicated CRLs. We initially excluded 38 CRL Distribution Points (CDPs) utilizing LDAP URLs, as this protocol is not supported by the CA/Browser Forum Baseline Requirements (CSBR, which mandates HTTP support). Subsequently, we attempted to retrieve the remaining CRL files via HTTP, successfully downloading 371 of them. The reasons for the retrieval failures are detailed in Table VI. The majority of failures were caused by CAs no longer providing the CRLs (maintenance lapsed).

Failures to retrieve some CRLs did not materially affect revocation-status queries for most certificates: 97.27% of the 23,252 certificates have accessible CRLs, enabling verification of whether the certificate had been revoked. This is because CRLs are highly centralized: many certificates issued by the same CA share the same CRL endpoint. Among the CRLs we obtained, the 50 largest lists (by number of entries) cover 92.45% of all certificates. In other words, most revocation checks rely on a relatively small set of widely used CRLs, so failures tend to affect only a long tail of CRLs that cover relatively few certificates rather than the bulk of our dataset.

D. Error code of SignTool

The error codes and corresponding messages returned by the SignTool signature validation are illustrated in Table V.

E. Refine Cluster Method with LLM

This section describes the methods we used to determine whether the subject CN fields of two certificates are similar during the certificate association process.

Method design. To this end, we introduce the state-of-the-art natural language processing method, Large Language Model (LLM), to achieve more accurate CN association recognition. Our method is named LLM for Certificate Clustering (LLM4CC). We employ GPT-4 [47], one of the top-performing LLMs currently, and carefully construct prompts to guide its recognition based on prior research findings and observational experiences.

TABLE IV: Impact of c^*_{mal} and r^*_{mal} on the labeled dataset of abused samples and certificates.

Types of Abuse	Total		Malware	
	# Samples	# Certs	# Samples	# Certs
Forged Signature (T1)	(+0.44, -0.22)	(+1.78, -1.44)	[-4.47, +3.22]	[-1.20, +3.14]
Certificates from Public CAs	(+0.43, -0.22)	(+1.86, -1.53)	[-4.62, +3.23]	[-1.14, +3.44]
Steal Certificate Key (T2)	(+0.58, -0.31)	(+4.23, -1.41)	[-4.47, +3.31]	[-1.26, +3.36]
Steal Developer ID (T3)	(+1.40, -0.55)	(+3.63, -1.55)	[-4.98, +3.41]	[-1.09, +4.35]
Fake Developer ID (T4)	(+1.69, -0.47)	(+3.20, -1.60)	[-2.52, +2.43]	[-0.88, +1.75]
Unspecified	(+0.44, -0.21)	(+1.80, -1.52)	[-4.62, +3.24]	[-1.15, +3.44]
Untrusted Certificate (T5)	(+0.29, -0.43)	(+1.39, -1.95)	[-3.61, +4.37]	[-0.99, +4.25]
Total	(+0.43, -0.22)	(+1.82, -1.46)	[-4.49, +3.23]	[-1.15, +3.30]

[1] $(\Delta_{c^*_{\text{mal}}=2}, \Delta_{c^*_{\text{mal}}=4})$ denote the relative changes (in %) in the total numbers of samples and certificates when c^*_{mal} is set to 2 and 4, compared to the baseline configuration (3).
[2] $[\Delta_{r^*_{\text{pup}}=0.4}, \Delta_{r^*_{\text{pup}}=0.6}]$ denote the relative changes (in %) in the numbers of malware samples and certificates when r^*_{pup} is set to 0.4 and 0.6, compared to the baseline configuration (0.5).

TABLE V: Error codes from SignTool verification.

Abusing Type	Error Code	Message
T1	-	No signature found
	0x80096010	The digital signature of the object did not verify
T5	-	A certificate chain processed, but terminated in a root certificate which is not trusted provider
	0x800B010A	A certificate chain could not be built to a trusted root authority
T2∼T4	-	Valid
	0x800B010C	A certificate was explicitly revoked by its issuer
	0x800B0101	A required certificate is not within its validity period

TABLE VI: The reasons for CRL retrieval failures.

Failure Reason	Description	Count
HTTP 502	The CA no longer provides this CRL	91
HTTP 404	The CA no longer provides this CRL	9
HTTP 403	The CA no longer provides this CRL	7
Others	CRL misconfiguration, etc	15
Total	CRL file not retrievable	122

Requirement: Below, I will input the names of two companies or individual entities as strings. Please help me determine if the two entities are the same.
Principles: 1) Ignore common substrings in the name that indicate company status, such as "LLC", "Co.", "OOO", "Inc" etc, and focus only on meaningful strings. 2) Overlook non-alphabetic characters in the name, such as punctuation marks like ".", ",", etc. 3) Ignore the case of letters and the singular/plural forms.
Criteria: 1) Some confusing letters in the words are acceptable. For instance, "Google" and "GooIe" or "G00gle" can be viewed as the same entity. 2) If two strings have a few differences, but the semantics are obviously similar, they can also be regarded as the same entity, even if they might not be in the same language. 3) If removing a part of one string does not affect its overall meaning and it then becomes identical to the other string, it can also be considered the same entity. 4) If the meaningful substring is a rare word, then it can be considered the same entity even if there are only one or two letters differing.
I will list some pairs of strings in the format of A@B below. For each pair, you only need to answer with "Y" or "N". "Y" means the two are the same entity, and "N" means otherwise.

The specific steps are as follows. We extracted the CNs from the certificates and paired them as input. Using the NED and PEM methods, we obtained 8,262 and 7,823 qualified CN pairs, respectively. The results from these two methods were then merged and input into LLM4CC for refinement. LLM4CC would respond whether the paired combination meets the matching criteria. Specifically, we utilized the GPT-4 API, entering the following prompt in the content with the role "system", and inputting the string pairs to be evaluated (Certificate Subject Name CN field) in the content with the role "user". Ultimately, we obtained 8,110 CN pairs.

Evaluation. We evaluate the effectiveness of LLM4CC and compare it with the two existing methods in prior work, NED [30] and PEM [31], from both design and experimental perspectives. First, from the design level, existing methods that rely on rule-based matching or exact matching have relatively weak scalability. We compared the capabilities of different certificate association methods in Table VII. Here, ●means that the certificate association method can handle the obfuscation strategy related to certificate polymorphism, while ○indicates the opposite. Our proposed LLM4CC method is capable of covering a wider variety of cases. We also evaluate those methods from the experimental level. We applied NED, PEM, and LLM4CC to our dataset, identifying 8,262 string pairs deemed similar by at least one method. Then we used manual review to check their effectiveness. Manual inspection revealed that LLM4CC demonstrated remarkable precision, with a random sample of 300 perfectly aligned with manual annotations. Then, we manually examined the parts where the three methods differed. We found that PEM's correct identifications were a subset of LLM4CC, and NED produced

TABLE VII: Comparison of different certificate association methods.

	Abbreviation Replacement	Case Replacement	Punctuation Change	Word Segmentation	Visual Confusion
NED	○	○	●	●	●
PEM	●	●	●	●	○
LLM4CC	●	●	●	●	●
	Reversing Word Order	**Synonym**	**Singular-Plural Change**	**Latin Letter Transcription**	**Special Character**
NED	○	○	●	○	●
PEM	●	○	○	○	●
LLM4CC	●	●	●	●	●

152 false positives (e.g., Shang Hai Zi Wei Wang Luo Ke Ji You Xian Gong Si vs. Shang Hai Shen Wei Wang Luo Ke Ji You Xian Gong Si). Besides, considering the potential issue of randomness in LLM output, we also evaluated the stability of LLM4CC, and found a 99.66% consistency over three rounds of independently repeated tests. In summary, we believe the method of identifying similar common names in this paper outperforms existing ones.

F. Method and Evaluation for Labeling Unspecified Certificates

We discovered that certificate association aids in categorizing unspecified certificates in previous steps. After applying the approach in Section III-B, we obtained 424 certificates labeled as T2∼T4. Clustering revealed that these belong to 379 clusters, each with a unique abuse type, indicating holders typically use a single abuse method. Then we can categorize one unlabelled certificate as the same abuse type as other labeled certificates within the same cluster. Such expansion draws not only from statistical observations but also adheres to the principles of code-signing abuse. For instance, if a certain certificate's public key is stolen, other certificates using this key can be considered stolen. Similarly, if a certificate is deemed as a fake ID, all certificates applied by that holder using that ID (i.e., certificates with the same CN name or key) should be classified as Fake ID. To reduce potential false positives, we only apply extended tagging to certificates that have signed malware. This method enhances the labeling method in Section III-B and has led to the identification of 287 previously unspecified certificates (159 Steal Certificate Key, 65 Steal Developer ID, 63 Fake Developer ID). Table I presents the final results incorporated with these 287 certificates.

In summary, we adopt two steps to categorize abuse types of certificates. First, we leverage the revocation reasons for labeling, drawing from the revocation information published by corresponding CAs. Although CA-provided information cannot be guaranteed to be 100% accurate, it remains the most authoritative reference for identifying abuse types. Second, we employ the association method for extended labeling, operating under the assumption that certificates held by the same attacker typically exhibit the same type of code signing abuse. To evaluate this assumption, we apply it to our labeled dataset from the first step. We divide the labeled dataset into training and testing subsets at different proportions, and the validation results are presented in Table VIII. We find that its accuracy is high, but the recall capability strongly depends on the size of the already-labeled training dataset. In this paper, we use this method to label another 287 certificates based on 157 labeled certificates.

TABLE VIII: Evaluation of certificate association methods.

Train:Test	Coverage	Accuracy
1:9	84.41%	100%
2:8	88.74%	100%
3:7	91.14%	100%
4:6	92.57%	100%

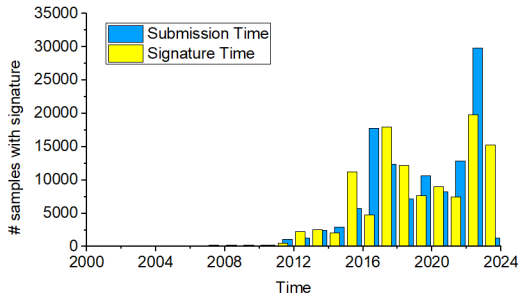

Fig. 11: Timestamp distribution of malware samples.

G. Samples distribution

We conducted a statistical analysis of the signing times provided by time stamping authorities of our collected malware samples, to better understand their distribution. The signing times are chosen in preference to compilation times stored in the "TimeDateStamp" field within the "IMAGE_FILE _HEADER" of the file header because, without valid timestamp signatures, this field can be modified by attackers to mislead security personnel or software and obscure malicious activities. We found a total of 116,783 malware samples had a valid signing timestamp, ranging from Mar. 1, 2001 to Apr. 23, 2024. According to the signing timestamps, the majority of our samples (78.25%) were issued after 2017. Therefore, analysis based on this data is expected to reveal new phenomena in the code-signing abuse ecosystem.

H. Requirements of CA/B Forum

Req.1 originates from Section 4.2, "Certificate Application Processing", of the CA/B Baseline Requirements [16].

Req.1 *The CA MUST also maintain and check an internal database listing Certificates revoked due to Code Signatures on Suspect Code and previous certificate requests rejected by the CA. CAs MUST not issue new or replacement Code Signing Certificates to an entity that the CA determined intentionally signed Suspect Code.*

I. Artifact Appendix

[Available badge] In this work, we conducted a large-scale measurement of code-signing abuse using signed malicious PE files collected from the wild. Through fine-grained classification, we identified a number of abused certificates and categorized them into five abuse types, creating the largest labeled dataset to date. To support open science, we release the above dataset and have filtered it to address ethical considerations. Our artifact is permanently archived at https://doi.org/10.5281/zenodo.17666996. Besides, the artifact can also be accessed at https://github.com/XingTuLab/Code_Signing_Abuse_Dataset.

The artifact is a dataset of **revoked** certificates used to sign **malware**. As described in Section "Ethics Considerations", we only release **malware-related** abused certificates, excluding PUPs, to avoid reputational harm to some legitimate developers. Considering the potential security risks of publishing all certificates, as unrevoked certificates and signed samples might be exploited by attackers, we release only confirmed **revoked** abused certificates, whose signatures can no longer pass client-side verification and thus pose no threat. In addition, since the signed software samples are commercially sensitive, we do not provide the original files but instead include URLs to their corresponding VirusTotal analysis reports.

TABLE IX: Description for abused certificate CSV file.

Field	Description
Cert MD5	MD5 hash of the abused certificate.
Serial	Serial Number of the abused certificate.
Subject	Subject Common Name (CN) of the abused certificate.
Issuer	Issuer Common Name (CN) of the abused certificate.
Country	Subject Country (C) of the abused certificate.
Valid From	"Not Before" timestamp of the abused certificate.
Valid To	"Not After" timestamp of the abused certificate.
Sample	URL to the VirusTotal report for a representative malware sample signed with the abused certificate.
Abuse Type	Labeled abuse type of the abused certificate.

The artifact contains a CSV table and a ZIP folder of certificate files. The CSV file mainly records metadata of abusive certificates—such as hash, serial number, subject, issuer, validity period, and abuse type—and provides the VirusTotal report of one representative software sample signed by each certificate, as shown in Table IX. The ZIP folder contains the original .cer files of all abused certificates listed in the CSV. Each file is named after its MD5 value, and the total number of certificates (2,072) is consistent with the description in Section IV-C of the paper.