Evaluating Security Checks Against Malicious Payloads with Forged Signatures

Lalchandra Rampersaud
Florida International University
lramp004@fiu.edu

Behzad Ousat
Florida International University
bousa001@fiu.edu

Seyed Ali Akhavani
Northeastern University
sadatakhavani.s@northeastern.edu

Javad Zandi
Florida International University
jzand@fiu.edu

Selcuk Uluagac
Florida International University
suluagac@fiu.edu

Amin Kharraz
Florida International University
mkharraz@fiu.edu

***Abstract*—Adversaries increasingly leverage diverse techniques to distribute malicious payloads across the web. One common, low-cost tactic is the hijacking of digital signatures and their attachment to malicious binaries, with the intent of deceiving both web browsers and operating systems. While code-signing certificates have traditionally served to verify the authenticity and integrity of software, adversaries now exploit these same certificates to evade detection mechanisms and facilitate the propagation of malicious code. This study seeks to empirically evaluate how modern web browsers respond to untrusted code by analyzing their reactions to signed malicious binaries. Our analysis shows that browsers' responses to certificate abuses may differ significantly, and the operating system may respond ineffectively potentially leaving end-users vulnerable to straightforward adversarial tactics. We also show that it is possible to significantly reduce the attack surface against certificate abuse with the use of a browser extension.**

***Index Terms*—malicious downloads, invalid/forged signatures, codesigning hijacks, browser/OS security**

I. Introduction

Today's software distribution ecosystem relies heavily on digital signatures to establish trust between users and the applications they install. Code-signing certificates, originally intended as a simple means of verifying publisher authenticity, have become central to the software supply chain by assuring users that downloaded binaries originate from legitimate sources [4], [5], [21], [25], [29], [37], [56], [75]. Yet, this very mechanism has increasingly been exploited by adversaries who obtain, forge, or hijack certificates to cloak malicious payloads under a veil of legitimacy. Such rogue signatures allow malware to bypass traditional security checks and remain undetected for longer periods, effectively undermining the foundation of trust on which modern binary verification rests.

This paper aims to investigate how modern web browsers, with the help of operating systems, shield users from digitally signed malware payloads distributed on the Web. This is an important question, as digitally signed malware payloads are increasingly being used to simulate legitimate payloads [26], [29], [72], and distribute over the internet.

This research aims to study how current security checks enabled by the web browsers or the operating systems handle signed binaries. Prior work has investigated several aspects of the code signing process and possible evasion techniques [4], [5], [29], [30], the certificates revocation [9], [30], and signing code [5]. However, the browsers' responses to such abuses are not that evident. We believe this is a critical question because interacting with web browsers constitutes a major part of the daily activities of millions of users, and the way browsers react to possible abuses if users are exposed to them plays an important role in the security posture of Internet users.

In particular, the paper first seeks to assess how browsers respond when encountering an improperly signed binary. This is an important step, as adversaries often leverage signed binaries to distribute malicious code covertly, reducing the likelihood of detection when users download them [4], [5], [29], [30]. The paper then evaluates the effectiveness of security features offered by the operating system to protecting users against improperly signed binaries. Finally, it seeks possible mitigation strategies to assist with reducing the attack surface with respect to improperly signed malware.

To answer these questions, we built an analysis pipeline by extracting certificates from 3,111 signed malicious binaries from 342 malware families. The extracted certificates belonged to 859 certificate issuers, including DigiCert (25.14%), Sectigo (9.13%), and GlobalSign (3.76%). The abused certificates belonged to various institutions, from software development companies such as Microsoft (8.52%), ESET (3.37%), and Nvidia Corporation (2.06%) to financial institutions. To emulate a real-world scenario, we built a dataset of more than 47K binaries and used the extracted signatures to generate improperly signed payloads. The collected binaries were designed to call functions similar to trojans, backdoors, and ransomware. Then, we developed a pipeline to simulate user interactions by hosting the payloads on servers and automatically retrieving and saving each file from the chosen browsers to assess the browsers' respective responses. In the following, we describe the main findings of the paper.

The security responses to forged signatures vary significantly. The experiment on the three popular web browsers shows that they respond differently to downloading files with

rogue signatures. For instance, in 42,269 of the files with improper signatures, Google Chrome allowed the download of payloads without any notifications. While Mozilla Firefox enforced a loose security policy on such payloads and allowed downloading all the payloads across different payload categories (See Table IV), Microsoft Edge blocked all the suspicious payloads.

Signed binaries with hijacked signatures have a relatively high chance of being downloaded without triggering any alarms compared to unsigned binaries. In particular, we observed that signed binaries with suspicious functionalities were downloaded successfully, similar to unsigned or properly signed binaries. Our static and dynamic analysis of major browser source code reveals that their validation mechanisms rely primarily on the reputation of the payload, while the authenticity of digital signatures is not thoroughly verified before permitting file downloads. Similarly, operating systems—the primary line of defense—does not enforce validation consistently at download time. This gap significantly increases the risk to user security and privacy by creating opportunities for successful malware delivery over the web.

We understand that defending against abused signatures requires a coordinated effort across different parties, such as certificate authorities, browser vendors, operating system vendors, and anti-malware companies. However, it is possible to significantly reduce the attack surface by using a lightweight approach to notify the user about the potential risk. To this end, we developed a browser extension that checks the certificate and signature used and notifies the user if an error is noted (see section IV).

We open-sourced [1] the entire evaluation pipeline, including 3,111 malware samples, 1,648 hijacked certificates, and extension source code.

II. Background and Related Work

Code signing is an essential practice in software security, ensuring authenticity and integrity within the digital ecosystem. Figure 1 illustrates the code signing and verification steps. While there are documents on CSP or mixed-content, which directly influence the file download procedures, we are unaware of any W3C standardization, specification, or documents that explicitly highlight how web browsers should operate when dealing with binaries with suspicious signatures. Here, we describe the high-level validation process:

1) Certificate and Timestamp Validation: This includes verifying the certificate's validity, certificate authority (CA), and whether the file timestamp is within the certificate validity period.
2) Hash Extraction: The signature section is decrypted with the certificate's public key, revealing the signer's original hash value.
3) Hash Calculation: The file's contents, excluding the digital signature, are hashed using the same algorithm used in the signature creation.
4) Hash Comparison: The extracted hash is compared to the newly calculated hash of the Portable Executable (PE) file. If they match, the file is considered unaltered since signing.

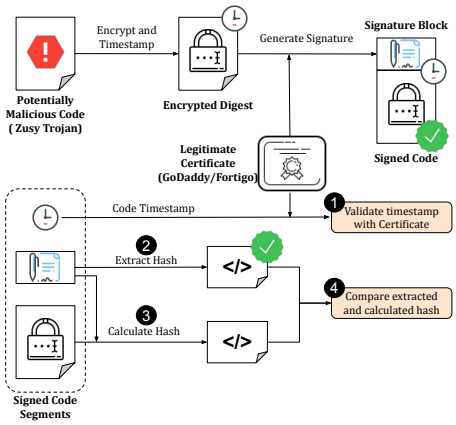


Fig. 1: A real-world example of delivering malware (Zusy Trojan) using a hijacked certificate.

A. A Motivating Example

There is no lack of evidence that adversaries use code signing certificates for their malware to exploit the trust mechanisms inherent in modern operating systems and software distribution platforms [4], [26], [34], [38], [56], [72]. In particular, code signing certificates are typically issued by trusted Certificate Authorities (CAs). When malware is signed with a valid certificate, it appears to be from a legitimate source. This can potentially facilitate bypassing built-in security measures to prevent untrusted code from being distributed through official channels. Figure 1 shows an example of how users are exposed to signed malicious binaries. In this example, in particular, the file could be a signed sample of Zusy Trojan. Since GoDaddy issues the certificate with Fortigo Inc., the signed code seems legitimate and does not raise any concerns. Our observations in major web browsers show that a range of incorrectly signed binaries are being downloaded successfully. Note that the example we discussed here is not synthetic and is a simplified version of this specific malware delivery mechanism using the web browser. Our experiments, discussed at length in Section IV, show that web browsers may react differently and may not prevent or notify users about the untrusted code.

B. Threat Model

In this paper, we assume that adversaries will use common techniques to distribute their malicious code and expose users. For instance, adversaries may lure users to malicious websites through social engineering, spear-phishing, or automatic redirection. The signed malware can be inserted into compromised or poorly regulated software distribution sites or third-party download platforms. The signed malware may appear as a legitimate update or software download, deceiving both the browser and the user.

[1] https://anonymous.4open.science/r/browserdownloadscheck-3306

We assume adversaries use common malicious practices in crafting their malicious payload to bypass current defense layers. For instance, malware authors may steal or purchase compromised code signing certificates, allowing them to sign malware and distribute it as legitimate software over the web. They could also forge certificates or exploit weaknesses in outdated cryptographic signing protocols. The paper focuses on scenarios where adversaries' effort to distribute malicious code is fairly small. That is, they distribute their code by building a catalog of benign software binaries, hijacking their signature blocks, and using them to modify and then distribute their malicious code. This method uses forged/invalid signature artifacts obtained from legitimate binaries. This sounded like a very cost-effective approach for adversaries. Our study aimed to analyze this matter in terms of effectiveness, impact quantitatively, and cost.

C. Related Work

The analysis of certificates, their adoption in the wild, and misissuance have been discussed in prior work. In this section, we discuss more relevant papers and elaborate more on the gap. Table I shows the summary of the gap analysis. Prior research on certificates is categorized into three high-level groups: (1) Certificate Abuses, (2) Stale Certificates, (3) Certificate Transparency & Verification.

Certificate Abuse. Certificate abuse refers to incidents where the digital certificates are exploited or misused with the intention of undermining security mechanisms [4]. misleading users, and facilitating cyberattacks [56]. This implies an erosion of trust, which relies on the authenticity of certificates and makes users skeptical of legitimately issued certificates. Certificate abuse can occur in several ways. For instance, SSL/TLS abuses often target verification mechanisms [17], [31], [36].The issue of abuse of code signing is also significant and is studied extensively [4], [5], [23], [24], [26], [34], [35], [38], [42], [56], [66], [72], especially where malware is signed using legitimate certificates obtained through theft, misrepresentation, or forgery.

Stale Certificate. Issues concerning revocation, long-lived certificates, and dangling records are all examples of what are referred to as stale certificates. The role of proper revocation mechanisms has been studied in prior work by showing the risk of invalid certificates [9], [45], [67], [77], [78] as well as techniques to improve the security and dependability of certificate revocation [9], [30] Other works have focused on providing faster update cycles for Certificate Revocation List (CRL) and better integration of revocation checks into software and systems that validate certificates [7], [32], [41], [45]. A centralized system for maintaining long-lived centralized systems for storing certification revocation information has been proposed in Korzhitskii et al. [32] as well as a mechanism for quick dissemination across different platforms and browsers. Dangling DNS records [7], [45] are obsolete or inaccurate DNS information pertaining to the domain names linked to SSL/TLS certificates originating from disuse or unauthorized access. Stale DNS entries might lead to complications in certificate validation [41].

Transparency and Verification. Transparency is key to preventing fraud in certificate issuance, promoting accountability, detecting misissuance, deterring malicious activity, and ensuring compliance with standards [26], [31], [34], [46]. A transparent process allows for independent verification and auditing, making Certification Authorities (CAs) accountable for following rigorous verification protocols before issuing certificates. To enhance verification, Kinkelin et al. [31] proposed a transparency architecture using distributed ledgers and smart contracts to implement multi-party validation in certificate signing. In line with other efforts, updated local certificate stores can play a crucial role by enabling revocation, removing compromised certificates, and adding new trusted root certificates [44]. Declining trust in public key infrastructure was studied through weak key analysis in SSL/TLS ecosystems [13]. Missing or erroneous certificate data further undermines trust [23], [37], [46], with Ma et al. [46] highlighting inaccurate identity information due to relaxed naming conventions and ownership changes.

Gaps in the State-of-the-Art. This paper aims to investigate a fundamental question of how browsers' policies are implemented to deal with evasive malware distribution by abusing the code signing process. Although substantial research has been conducted on topics such as signed malware, their distribution mechanisms [26], [35], and evasion techniques [14], [16], [57], our understanding of how browser-level security mechanisms are configured to shield users from these risks is still abstract. This paper seeks to fill this gap by providing an empirical analysis of browser responses to certificate abuse, evaluating their policy on improperly signed binaries, and exploring potential enhancements to improve user protection in the face of increasingly sophisticated attacks.

III. Methodology

In this section, we elaborate on the research questions. We describe how to construct a dataset and build the experimentation pipeline.

A. Research Questions

We aim to evaluate the response of modern browsers to signed binary payloads that are not necessarily seen in the past (i.e., Safe Browsing and other reputation-based services have not listed them as malicious), signed with hijacked signatures obtained from legitimate entities. We analyze the use of invalid/forged signatures, not valid stolen certificates. In light of the goal, our study attempts to respond to the following research questions:

RQ1: The Validation Process of Code Signing in Browsers and its Effectiveness: We analyze how browsers respond differently when presented with a malformed signed executable. In particular, we aim to check how current mechanisms perform signature validation and what browser is more likely to allow malformed signed executables. We also analyze the built-in security features of modern web browsers in blocking and isolating executables with malformed signatures.

TABLE I: Prior research on certificate abuse, transparency, and verification. This paper aims to augment prior work by analyzing how web browsers shield users against signed malware in the wild.

Venue	Abuse	Stale Certificate	Transparency & Verification	Dataset	Analysis Method
IoT'24 [26]*	Forgery/Key Security/Trust	-	Identity Fraud	81 APT CVE from the ATT&CK database	Code signing process protection
PAM'24 [9]	-	Revocation		1.5M cert replacements	Preventing post-revocation usage in certs
NOMS'24 [78]	-	Revocation	-	5.2M revoked certs across 2M orgs	Revocation checking popular browsers
SecDev'23 [5]*	Key Security	-	-	No dataset	Analysis of code signing in CI/CD
IMC'23 [45]	-	Dangling DNS/Revoke	-	5B certificates analyzed	Cross-referenced revocation info with certs
TrustCom'23 [42]*	Cert Verif	-	-	5.5M malware samples from VT	Response from verify tools: Sigcheck, SignTool
SIGSOFT'23 [24]*	Key Security	-	-	No dataset	Partial hashing of components to detect alterations
Yi'22 [77]*	-	Revocation	-	No dataset	Protocols using ledgers for revocation records
USENIX'21 [46]	-	Long-Lived Certs	Inaccurate Cert Data	2.9B certificates	Ownership and control of CA certificates
PAM'21 [32]	-	Revocation	-	1.7M certificates from Censys dataset	CRL and OCSP statuses from expiration to removal
IMC'21 [44]	-	-	Outdated Cert Store		
DFRWS'21 [72]*	Trust abuse	-	-	9 signed malware from public repos	Used sigcheck to analyze memory dumps
Concordia'20 [23]*	Cert.Verif	Revocation	Inaccurate Cert Data	79+K applications from download portals	Analyzed the signatures of each file
NDSS'20 [67]	-	Revocation	-	84.1M certificates from Censys.io	Analyzed revocation schemes
NOMS'20 [31]	Cert.Verf		Identity Fraud		
SIGSAC'19 [40]			Cert Dissemination	3+M certs from 6K Alexa Top-1M websites	Effectiveness of cert transparency monitors
S&P'18 [37]	-	-	Inaccurate Cert Data	240M browser-trusted certs in Censys	Analyze how well CAs construct certificates
USENIX'18 [30]*	-	Revocation	-	145+K code signing certs from Malsign, VT, Wine	Compromised certs and revocation delays
WEIS'18 [35]*	Illicit Trade	-	-	14+K signed malware from VT with 1,163 certs	Tracking of code signing certs to black-market vendors
IMC'18 [65]	-	-	Certificate Privacy	Probe outgoing connections at UCB	Deployment of CT via network monitoring
NDSS'17 [38] *	Trust Abuse	-	-	33.3M download events from WINE.	Detect silent delivery campaigns and commands.
PKIA'17 [56]*	Trust Abuse	-	-	No dataset	Impact of signed malware
WWW'17 [36]	Cert.Verif	-	-	Load the root page for 944K sites	Impact of trusting third-party content via HTTPS
WWW'16 [4]*	Trust Abuse	-	-	3.3 million samples with 800k certs	Digital cert characteristics.
IMC'16 [10]	Forgery	-	Incorrect Cert Data	80+M certs from IPv4 port 433 scans	Prevalence of invalid SSL certs.
CCS'15 [34]*	Trust Abuse	-	Identity Fraud	356K samples from VirusShare, CCSS	Signed PUP or Malware. CA defenses and revocations.
S&P'14 [22]	Forgery	-	-	3+M certs from Facebook	Determine authenticity of certs
ACM Queue'14 [39]	-	-	Identity Fraud/Cert Dissemination	No dataset	Analyze alternatives to the CA
IMC'14 [79]	-	Revocation	-	943+K certs taken from scans of IPv4 port 433	Reissuing and revoking certs post Heartbleed.
Viitattu'2013 [66]*	Key Security	-	-	No dataset	Best practices to avoid a breach.
IMC'13 [13]	-	-	Weak keys	42+M certs from 108+M IP addresses	Study of the HTTPS certificate ecosystem.
CCS'12 [17]	Cert.Verif	-	-		Implementation of SSL libraries in common software

RQ2: Safeguarding and Attack Surface Management: We investigate how browsers protect users from previously unknown signed binaries. What would be the role of in-browser mechanisms when payloads are not listed in block lists? How can current methods (e.g., Safe Browsing) be augmented with minimal changes to the code or the logic of the browser functionalities to enhance browser trustworthiness against sophisticated malware distribution strategies in the wild?

B. Dataset

In this section, we describe how we generated the dataset for this experiment by collecting signed malware samples and using their certificates to analyze browsers' responses. The steps taken are as follows:

Collection of Malware Samples: Building a dataset of real-world signed malware is a non-trivial task. To collect these samples, we automatically fetched malware samples daily from popular public malware repositories such as MalShare [47], VirusTotal [74], and GitHub Malware dumps [69]. We collected 79,544 malware samples and identified 3,111 signed malware samples that belonged to a variety of families, including known Remote Access Trojans (RATs) [15], [27], [76], Potentially Unwanted Programs (PUPs) [33], [34], [71], and Ransomware [28], [55], [63]. Table II shows the distribution of the signed malware, including the number of families and their occurrences and variations collected in the wild.

Extraction of Unique Certificates: We proceeded to extract the digital certificates from the payload header utilized for signing the malware samples. This was achieved using the `X509Certificate2` class from the .NET 8 Framework, where the signed binary is loaded to create a `X509Certificate2` object that is then used to export the certificate to disk. This is possible because Authenticode [49] is the code signing technology used to sign PE files on Windows. It is based on the X.509 standard certificate [61] and the PKCS #7 file structure standard [49], the signature used in signing PE files. Although PKCS #7 structure is standard, there are several differences between X.509 certificates, such as their type and intended use, extensions, signature algorithms, and public key types. These variations enable them to be customized for various security requirements, including identity verification, code signing, and TLS/SSL for web communication. This produced 1,648 distinct instances from

*Studies that concentrate on code signing certificates. All others are geared at SSL/TLS.

TABLE II: Distribution of signed malware samples

Malware	Families-Occurrences	Variants
Ransomware	**8 - 23 (0.74%)**	
LockScreen	13 (0.42%)	1
StopCrypt	3 (0.10%)	3
LockBit	2 (0.06%)	1
Others	5 (0.16%)	5
RAT	**235 - 2,755 (88.55%)**	
Wacatac	262 (8.42%)	2
Leonem	210 (6.75%)	1
Sabsik	210 (6.75%)	1
AgentTesla	196 (6.30%)	59
GuLoader	163 (5.23%)	1
RedLineStealer	264 (8.49%)	1
Tnega	123 (3.95%)	1
Woreflint	43 (1.38%)	1
Others	227 - 1,868 (60.05%)	669
PUP	**100 - 320 (10.28%)**	
InstallCore	36 (1.16%)	2
GamingRelatedHytekModLoader	24 (0.77%)	1
LoadMoney	18 (0.58%)	1
DownloadGuide	15 (0.48%)	1
Razy	15 (0.48%)	2
Others	94 - 212 (6.81%)	11
Self Replicating Malware	**10 - 13 (0.42%)**	
VBInject	2 (0.06%)	1
Viking	2 (0.06%)	1
Others	8 - 9 (0.30%)	9
Total	**342 - 3,111**	**895**

1,270 entities, including Microsoft, Nvidia Corporation, Brave, and ESET Internet Security. Table III shows the top 804 (25.84%) of abused organizations' certificates taken from the signed malware. After conducting a careful validation process, we identified 1,648 unique certificates from 1,270 entities, including Microsoft, Nvidia Corporation, Brave, and ESET Internet Security. The malicious binaries were analyzed using Windows signtool to check the validity of the certificates. In binaries with a hijacked signature, the output indicates that the hash is invalid according to the hash stored in the file's digital signature.

TABLE III: Most abused certificates among malware families

Subject	Instances	Top Malware Families
Microsoft Corporation	265 (8.52%)	Smokeloader,Wacatac Sabsik
ESET	105 (3.37%)	RedLineStealer, AgentTesla, FormBook
Wen Jia Liu	70 (2.25%)	AgentTesla, FormBook, Leonem
Google	65 (2.09%)	AgentTesla, Sabsik, Wacatac
Nvidia Corporation	64 (2.06%)	RedLineStealer, Sabsik, Tnega
Philandro Software GmbH	51 (1.64%)	FormBook, AgentTesla, AveMariaRAT
Blanchard Willy	43 (1.38%)	GamingRelatedHytekModLoader, msil, Wacatac
The Tor Project, Inc.	38 (1.22%)	FormBook, AgentTesla, Leonem
Ivan Yurievich Permyakov IP	36 (1.16%)	RedLineStealer, Sabsik, Wacatac
Gary Kramlich	34 (1.09%)	RedLineStealer, Sabsik, Stealer
Adobe Systems Incorporated	33 (1.06%)	AgentTesla, Casdet, Malgent
Others	2,307 (74.16%)	Wacatac, Leonem, RedLineStealer

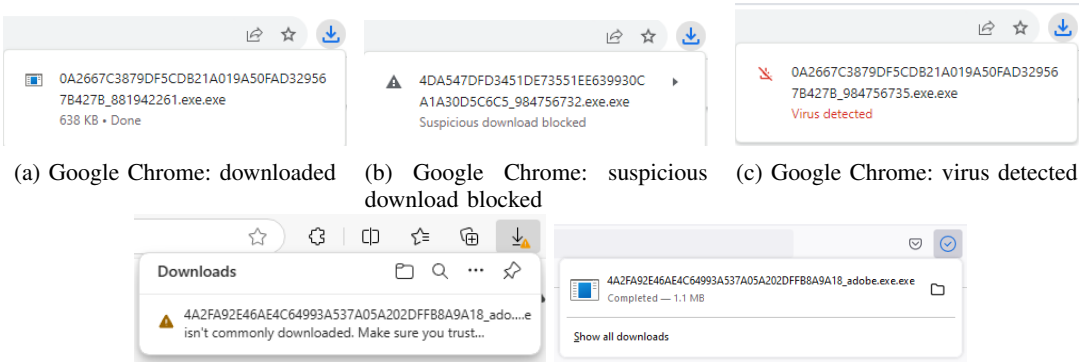
Building test-cases for the large-scale test. To conduct a large-scale experiment, one approach involves downloading real-world signed malicious binaries and simulating user browsing sessions to retrieve files via web browsers. However, this experiment presents non-trivial challenges. While obtaining a substantial number of signed malware samples can pose significant challenges, the primary difficulty lies in the fact that many of these files may already be flagged, irrespective of their signed status. Consequently, it is difficult to measure if browsers respond to already-known hashes or malformed certificates. To address this, we define "patient zero" scenarios, wherein adversaries attach hijacked signatures to previously unknown samples where the file's hash has not yet been classified as malicious. To this end, we generated a large dataset of binaries that aim to provide core functionalities used by trojans and backdoors by establishing sockets and connections to C&C servers or requesting access to important systems services such as the microphone, camera, windows registry, Windows Management Instrumentation (WMI) APIs. The generated binaries were designed to mimic the structure and behavior of malicious files(e.g., establishing backdoors) but did not introduce any actual harmful payloads or functionalities. We utilized the previously extracted signatures to improperly sign the benign executable files. We affixed all 1,648 distinct signature headers to each of the benign files at the PE signature location, resulting in a substantial collection of PE files with invalid signatures. This phase is crucial in replicating the possible abuse of these certificates in real-world scenarios.

To simulate the behavior of hijacked signature blocks or improperly signed binaries, we deliberately included mismatches in the cryptographic hash or omitted certain validation data. The goal was not to produce cryptographically valid signatures but to test how three major web browsers (Microsoft Edge version 124.0.2478.51 64-bit, Google Chrome version 124.0.6367.60 64-bit, and Mozilla Firefox version 122.0.1 64-bit) respond when presented with binaries that included recognizable certificate chains but lacked proper cryptographic integrity. This is a common practice among malware authors, and our hypothesis was to test whether web browsers respond effectively to this malicious practice. We extracted digital signatures from signed Portable Executable (PE) files by reading the header and appending them to other files, effectively creating invalid signatures.

The generated binaries were divided into four categories: suspicious files, third-party, Windows System32, and Data processing. To create suspicious files, we created binaries with seemingly harmful abilities, such as system reconnaissance, which includes calling a Windows Management Instrumentation (WMI) method [18], [68], [73] to obtain general information about the processor, disk, and Operating System, Windows API calls such as Send [19], and unauthorized microphone access [43], [62]. Third-party files are obtained from legitimate sources such as Adobe PDF Reader (version 23) [1] and Python (version 3.12.4) [60]; Windows System32 files are taken directly from the System32 folder [53]. Lastly, the data processing files (Internal Business Applications) are binaries that perform basic data processing on the local machine with read/write requirements. The distribution of files signed by the 1,648 certificates are as follows: eleven suspicious files generating 18,128 signed instances, thirteen third-party applications generating 21,424 instances, two Windows System32 binaries generating 3,296 instances, and three data processing files generating 4,944 instances. All these generated files and the scripts are available in the project's public repository. The defined categories aimed to cover realistic scenarios relevant to binary classification in the security ecosystem, including distinctly malicious operations (suspicious) and completely legitimate applications (third-party and System32), as well as standard software functionalities (data processing).

Generalizability We considered three metrics to reduce potential biases and build a more generalizable dataset. First, the dataset contains several forms of modern malicious code in the wild. In particular, it contains samples from 342 signed malware families and 895 variants, delivering trojans, ransomware, cryptojacking, and potentially unwanted programs (PUPs). Second, the certificates being used are issued for 1,270 unique entities. Lastly, certificates have been issued by various certificate authorities located in Europe, the US, and Asia. While we considered these metrics, we still believe that our analysis could be impacted by other forms of biases. We should note that the process has been a best-effort approach, given all the limitations to acquiring real-world malicious datasets.

Browser Testing Experiment. We made the signed payloads available for download in a controlled environment in which



(a) Google Chrome: downloaded (b) Google Chrome: suspicious download blocked (c) Google Chrome: virus detected

(d) Microsoft Edge: suspicious download blocked (e) Mozilla Firefox: downloaded

Fig. 2: Browser responses to download of malformed signed binary files

we uploaded the signed files to a dedicated server. The server is configured to mimic a real-world web hosting environment, allowing for the testing of file downloads through various browsers. We automated the process of downloading the files and capturing the responses from various browsers. We used Selenium [8] version 4.15.2 to systematically download each file using different browsers (Google Chrome, Microsoft Edge, and Mozilla Firefox).We selected Selenium because of its compatibility with the leading browser vendors for running benchmarks. Other newer frameworks such as Playwright [52] might have provided more straightforward, built-in handling but are more geared towards testing web apps and not desktop applications [52]. Additionally, we found that Playwright did not enumerate the download process as would have been seen by the end user; it uses temporary files and no download is marked as possibly malicious in any browser. However, given maturity, wider browser compatibility, and familiarity, Selenium is still a popular and strong choice [2] for testing browser variability. We wrote an automation script to run web sessions and download the target files. It opened the browser to the file's URL, captured screenshots every 5 seconds during the download, and then closed the browser. The screenshots shown in figure 2 were analyzed to classify the browser's responses to the download attempts. Machine learning techniques were employed to process and analyze the screenshots and categorize the responses into distinct outcomes such as "virus detected," "suspicious download blocked," or "downloaded successfully." Specifically, the Google Chrome categorization was derived from the observable messages and actions exhibited by the browser throughout the download procedure, as seen in figures 2a, 2b, and 2c. An examination of the screenshots taken on file download was carried out to categorize the browser's reactions to the download attempts. This step involves using supervised learning to train ResNet50 [11], [20], a deep convolutional neural network model [3] to recognize and categorize the visual cues from the screenshots with an accuracy of 97.01%. The experimental pipeline is shown in figure 3.

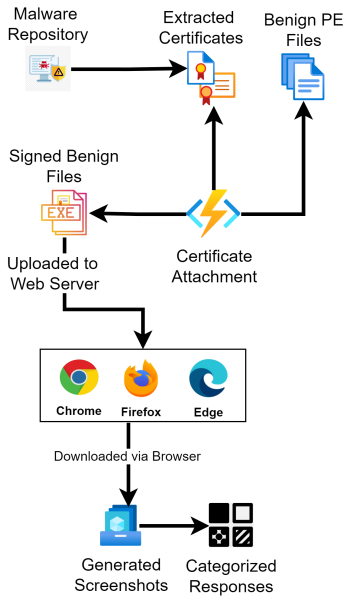


Fig. 3: The high level view of the experiment pipeline. The study consists of using signatures extracted from modern malware to sign benign files, which are subsequently downloaded from a website using popular browsers.

IV. Real-World Evaluation

A. Certificate Validation and Response Variability

This section investigates how modern web browsers handle binary validation and their effectiveness in managing payloads exposed to users. Specifically, by investigating the security mechanisms, we analyze how browsers inspect binaries, the variations in their choice of security integration, and their differing responses to files with various certificates.

Analysis of Browser Mechanisms The results show distinct differences across the three browsers for our suspicious files. When using Google Chrome, 14,200 files were successfully downloaded, 3,906 were blocked, and 22 were flagged as "virus detected." Microsoft Edge blocked all files, while Mozilla Firefox allowed every file to be downloaded with-

TABLE IV: Browser responses while downloading signed binaries with hijacked certificates. While Microsoft Edge blocked all the suspicious cases, Google Chrome (partially)and Mozilla Firefox (fully) allowed the downloads.

	Google Chrome			Microsoft Edge			Mozilla Firefox		
File	**Downloaded**	**Blocked**	**Virus detected**	**Downloaded**	**Blocked**	**Virus detected**	**Downloaded**	**Blocked**	**Virus detected**
Suspicious	14,200 (78.33%)	3,906 (21.55%)	22 (0.12%)	-	18,128 (100%)	-	18,128 (100%)	-	-
Third-party	17,071 (79.68%)	3,742 (17.47%)	611 (2.85%)	-	21,424 (100%)	-	21,424 (100%)	-	-
Windows System32	2,678 (81.25%)	613 (18.59%)	5 (0.15%)	-	3,296 (100%)	-	3,296 (100%)	-	-
Data processing	3,967 (80.22%)	974 (19.70%)	3 (0.18%)	-	4,944 (100%)	-	4,944 (100%)	-	-

out warnings. This demonstrates variability in how browsers handle file downloads and their safety assessments despite similar core principles. Table IV presents the results, indicating whether files were successfully downloaded, blocked, or marked as containing a virus. We further investigated these results by analyzing each browser's security mechanism to explain the discrepancies in handling file downloads.

Microsoft Edge utilizes Microsoft Defender SmartScreen [12] as its primary security feature. SmartScreen checks URLs against a database of known phishing, malware, and scam websites to protect users. Additionally, it verifies file integrity using file hashes and digital signatures, using an extensive reputation database to assess the risk associated with downloads. Files lacking a reputation might trigger warnings such as "not commonly downloaded," advising users to proceed cautiously. Developers can avoid these warnings by signing files with trusted Authenticode Certificates issued by a Windows Root Certificate Program CA. SmartScreen relies on multiple indicators such as file popularity, download history, and originating URL to assess risks [51].

Google Chrome employs Google Safe Browsing [6], [58], [59] for download protection. To understand the variation in Google Chrome's results, we analyzed its' pipeline from file download initiation to disk finalization [58], [59]. When a file is downloaded, Google Chrome evaluates it by sending metadata (file hash, URL, and referrer chain) to Google Safe Browsing, which checks the file against its constantly updated database of known threats. This process helps classify the file as `safe`, `unsafe`, or `unknown`. While Google Chrome uses download protection based on Safe Browsing's decision, it does not typically prevent downloading improperly signed executable files. Additionally, Google Chrome uses local heuristics and periodically updates its databases with new malware signatures to enhance user protection [6], [64].

Listing 1 shows the data sent to the Google Safe Browsing service, while the corresponding responses are displayed in Listing 2. While this process explains the variability in Google Chrome's responses, the absence of file signature analysis during download is a notable limitation. We observed that Google Chrome did not consistently block all suspicious files during our tests. Google Chrome also offers real-time protection by frequently updating its local databases with new malware signatures and harmful URLs, accessible via `chrome://safe-browsing/#tab-db-manager` [64]. It is important to note that the signature information of downloaded files is not included in the request. When a file is flagged as a threat, the browser immediately warns the user, offering options to either delete the download or proceed at the user's own risk.

Mozilla Firefox uses Google's Safe Browsing service, similar to Google Chrome, to screen files and websites against known malware and phishing threats [54]. If a file is identified as suspicious, Mozilla Firefox us supposed to issue a warning advising the user of the potential danger [48]. However, unlike Microsoft Edge, Mozilla Firefox allowed the download of all files during our experiment, even when certain files were flagged as potentially dangerous by other browsers. Table V shows the Safe Browsing integration among all the studied browsers.

TABLE V: Safe Browsing Implementation Across Browsers

Browser	**Integration**	**Warning**
Google Chrome	Google Safe Browsing	Warning for suspicious files occurs before download, allowing the user to choose to accept the file. The virus-detected message is shown post-download, and the temporary file is deleted
Mozilla Firefox	Google Safe Browsing	Mozilla Firefox integrates Safe Browsing, but in our experiments did not issue warnings.
Microsoft Edge	Microsoft Defender SmartScreen	Warning for suspicious files are blocked pre-download, allowing the user to choose to download the file.

Further Analysis. Table IV provides key insights into how these browsers handled downloads of Portable Executable (PE) files. In summary, our large-scale experiment shows that while the fundamental principles of Safe Browsing are similar across browsers, their actual implementation and responses vary significantly. Microsoft Edge's strict enforcement through SmartScreen stands in contrast to Google Chrome's reliance on Safe Browsing metadata, and Mozilla Firefox's permissive behavior raises questions about the adequacy of its security protocols for signed binaries. The discrepancies in how these browsers handle the same binaries indicate that web users' security depends heavily on their browser of choice.

B. Safeguarding and Attack Surface Management

As a proof of concept to reduce the attack surface, we developed a browser extension as an augmented method to block payloads with a suspicious signature and notify the user about potential harm. The extension inspects and analyzes the payload using an out-of-band technique, with no changes to users' browsing experience. *WinVerifyTrust* [50], a Windows built-in API function was used to verify the authenticity and validity of signed binaries. This functionality can be found in the *wintrust.dll* binary [50] and uses a predefined data structure, *WINTRUST_DATA* to describe the binary being verified. This verification method was chosen because it is

Listing 1: Sample of the request sent to Google Safe Browsing containing the downloaded file information to be analyzed

```
{   file_basename : ne.exe , length : 7984, digests.sha256 : 9A7...0 ,
    population : {
      finch_active_groups : [ SafeBrowsingArchiveImprovements.Enabled ],
      user_agent : Chrome/124.0.6367.60/Windows , user_population : ENHANCED_PROTECTION
   },
    referrer_chain : [ {
       ip_addresses : [ 20.106.193.196 ], referrer_url : https://example.com/ ,
       type : EVENT_URL , url : https://example.com/notepad.exe }, {...}, {...} ],
    url : https://example.com/ne.exe
}
```

Listing 2: Sample response from Google Safe Browsing with the result of the file analysis

```
{
   token : 7a75470... ,
   verdict : SAFE
}
```

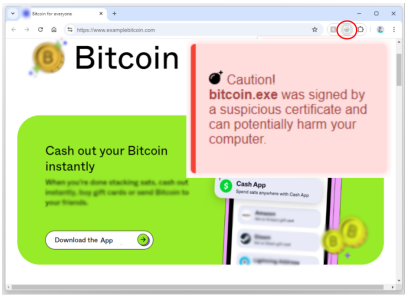


Fig. 4: Blocking the trojan with rogue certificate using the proposed extension

a mature high-level wrapper around complex cryptographic checks. The imposed overhead of using WinTrustVerify offline includess signature decoding and cryptographic validation. Benchmarking using the results from FastSigCheck [70] indicated that WinTrustVerify can perform signature verification in approximately 25 to 50 milliseconds, depending on the file's signature status. The extension ensures that payloads with known or unknown reputations cannot be downloaded if the certificate looks suspicious. This concept is shown in Figure 4, where the extension blocks a rogue certificate used to sign a trojan.

V. Discussion

This paper demonstrates that generating and deploying malicious payloads with a hijacked certificate is both low-cost and easily scalable for adversaries, increasing the chance of successful attacks against web users. Following our findings, we discuss the implications of our investigation and make recommendations for potential routes forward.

Shielding Web Users From Highly Evasive Web Threats. The core insight in this paper is that unknown payloads with suspicious certificates, even if they are not in the blocklists, should not be trusted. Users should be alerted about potential consequences so that they can make an informed decision. This is a critical step to reduce user exposure to highly evasive malware attacks and patient-zero threats in the wild. We should note that W3C does not explicitly have any standardization on fetching malicious payloads. While there are documents on CSP or mixed-content that directly influence the file download procedures, we are unaware of any standardization, specification, or documents related to malicious code download. That said, we argue that host operating systems should take a more proactive role as the primary layer of defense. Browser vendors are unlikely to assume the role of policing the Internet by independently determining the maliciousness of payloads, as doing so could lead to disputes and complaints from developers and content providers.

Call for More Effective Collaboration. The escalating threat posed by certificate abuse in signed binaries underscores the urgent need for stronger collaboration between key stakeholders, including operating systems and certificate authorities (CAs). Collaborative efforts enable the development of unified frameworks for certificate validation, revocation, and monitoring, significantly reducing the attack surface for adversaries who leverage abused or stolen certificates to bypass browser security. Furthermore, through enhanced cooperation with CAs, browser vendors could implement stricter certificate validation protocols, ensuring that all signed binaries undergo deeper scrutiny before being trusted. This might include real-time checks for certificate revocation, stronger enforcement of short-lived certificates, and enhanced integration of transparency logs that allow the tracking and auditing of certificates. That said, we should also note that finding the right incentives for all the involved entities, such as Web browsers, operating systems, and certificate authorities, might be challenging, as these entities might be wary of policing content on the web. However, a coalition in this direction to develop a standardized framework across browsers and Operating System (OS) vendors is critical.

VI. Conclusion

In this paper, we performed an empirical analysis of major browser responses to hijacked signatures. Our analysis revealed that browsers may respond differently when dealing with signed binaries using hijacked signatures. The source code analysis revealed that web browsers mainly focus on the reputation of the payloads.The variability in response to invalid signatures indicates the need for more standardized

security standards across browsers to offer consistent user protection regardless of browser choice. By improving these methods, browsers can better protect users from the hazards associated with compromised software, preserving confidence and integrity in the digital realm.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their thoughtful feedback. This project was supported by Microsoft AI Security, US National Security Agency (NSA) under Grant No. H98230-21-1-0324, US National Science Foundation (NSF) under Grant No. 2219920, and Intergovernmental Personnel Act Independent Research & Development Program. Opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Adobe. Adobe acrobat reader. https://get.adobe.com/reader/, 2024.

[2] Saad Almabruk, Samia Abdalhamid, and Tahani Almabruk. Comparative reliability analysis of selenium and playwright: Evaluating automated software testing tools. *Asian Journal of Research in Computer Science*, 18(1):34–44, 2025.

[3] Neena Aloysius and M Geetha. A review on deep convolutional neural networks. In *2017 international conference on communication and signal processing (ICCSP)*, pages 0588–0592. IEEE, 2017.

[4] Omar Alrawi and Aziz Mohaisen. Chains of distrust: Towards understanding certificates used for signing malicious applications. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 451–456, 2016.

[5] Pranshu Bajpai and Raghudeep Kannavara. Misplaced trust: The security flaw in modern code signing process. In *2023 IEEE Secure Development Conference (SecDev)*, pages 49–50. IEEE, 2023.

[6] Simon Bell and Peter Komisarczuk. An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–11, 2020.

[7] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. Cloud strife: mitigating the security risks of domain-validated certificates. 2018.

[8] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. Web application tests with selenium. *IEEE software*, 26(5):88–91, 2009.

[9] David Cerenius, Martin Kaller, Carl Magnus Bruhner, Martin Arlitt, and Niklas Carlsson. Trust issue (r) s: Certificate revocation and replacement practices in the wild. In *International Conference on Passive and Active Network Measurement*, pages 293–321. Springer, 2024.

[10] Taejoong Chung, Yabing Liu, David Choffnes, Dave Levin, Bruce MacDowell Maggs, Alan Mislove, and Christo Wilson. Measuring and applying invalid ssl certificates: The silent majority. In *Proceedings of the 2016 Internet Measurement Conference*, pages 527–541, 2016.

[11] Torch Contributors. Pytorch resnet50. https://pytorch.org/vision/0.18/models/generated/torchvision.models.resnet50.html, 2024.

[12] David Coulter Dan Wesley, Andrea Barr. Microsoft edge support for microsoft defender smartscreen. https://learn.microsoft.com/en-us/deployedge/microsoft-edge-security-smartscreen, 2024.

[13] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 291–304, 2013.

[14] Daniele Cono D'Elia, Emilio Coppa, Federico Palmaro, and Lorenzo Cavallaro. On the dissection of evasive malware. *IEEE Transactions on Information Forensics and Security*, 15:2750–2765, 2020.

[15] Aya El-Sayed El-Metwaly, Mahmoud Amr Abdelfattah, Nada Magdy Maher, Mohamed Hamed, Eslam Mohamed Tayel, Maryam Ashraf Al-Rifai, and Ali E Takieldeen. Remote access trojan (rat) attack: A stealthy cyber threat posing severe security risks. In *2024 International Telecommunications Conference (ITC-Egypt)*, pages 1–5. IEEE, 2024.

[16] Jiaxuan Geng, Junfeng Wang, Zhiyang Fang, Yingjie Zhou, Di Wu, and Wenhan Ge. A survey of strategy-driven evasion methods for pe malware: transformation, concealment, and attack. *Computers & Security*, 137:103595, 2024.

[17] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 38–49, 2012.

[18] Matt Graeber. Abusing windows management instrumentation (wmi) to build a persistent, asynchronous, and fileless backdoor. *Black Hat. Las Vegas, NV, USA*, 2015.

[19] Sanchit Gupta, Harshit Sharma, and Sarvjeet Kaur. Malware characterization using windows api call sequences. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, pages 271–280. Springer, 2016.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Hannes Holm and Erik Hyllienmark. Hide my payload: An empirical study of antimalware evasion tools. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2989–2998. IEEE, 2023.

[22] 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. IEEE, 2014.

[23] Mina Jafari. *Measuring the Effectiveness of Microsoft Authenticode: A Systematic Analysis of Signed Freeware*. PhD thesis, Concordia University, 2020.

[24] Ajay Jain and Kusha Chopra. Malware signing detection system. *ACM SIGSOFT Software Engineering Notes*, 38(5):1–8, 2013.

[25] Gheorghe Jescu. Can we trust a trustee? an in-depth look into the digitally signed malware industry. 2014.

[26] Tiantian Ji, Binxing Fang, Xiang Cui, Tian Wang, Yuntao Zhang, Fan Gu, and Chao Zheng. Scrutinizing code signing: A study of in-depth threat modeling and defense mechanism. *IEEE Internet of Things Journal*, 2024.

[27] Dan Jiang and Kazumasa Omote. An approach to detect remote access trojan in the early stage of communication. In *2015 IEEE 29th international conference on advanced information networking and applications*, pages 706–713. IEEE, 2015.

[28] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 12th International Conference, DIMVA 2015, Milan, Italy, July 9-10, 2015, Proceedings 12*, pages 3–24. Springer, 2015.

[29] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. Certified malware: Measuring breaches of trust in the windows code-signing pki. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1435–1448, 2017.

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

[31] Holger Kinkelin, Richard von Seck, Christoph Rudolf, and Georg Carle. Hardening x. 509 certificate issuance using distributed ledger technology. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6. IEEE, 2020.

[32] Nikita Korzhitskii and Niklas Carlsson. Revocation statuses on the internet. In *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29–April 1, 2021, Proceedings 22*, pages 175–191. Springer, 2021.

[33] 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.

[34] 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*, pages 465–478, 2015.

[35] Kristián Kozák, Bum Jun Kwon, Doowon Kim, and Tudor Dumitraş. Issued for abuse: Measuring the underground trade in code signing certificate. *arXiv preprint arXiv:1803.02931*, 2018.

[36] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J Alex Halderman, and Michael Bailey. Security challenges in an increasingly tangled web. In *Proceedings of the 26th International Conference on World Wide Web*, pages 677–684, 2017.

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

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

[39] Ben Laurie. Certificate transparency: Public, verifiable, append-only logs. *Queue*, 12(8):10–19, 2014.

[40] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. Certificate transparency in the wild: Exploring the reliability of monitors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2505–2520, 2019.

[41] Daiping Liu, Shuai Hao, and Haining Wang. All your dns records point to us: Understanding the security threats of dangling dns records. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1414–1425, 2016.

[42] Guangqi Liu, Qiongxiao Wang, Cunqing Ma, Jingqiang Lin, Yanduo Fu, Bingyu Li, and Dingfeng Ye. The broken verifying: Inspections at verification tools for windows code-signing signatures. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 804–813. IEEE, 2023.

[43] Yuchen Liu, Ziyu Xiang, Eun Ji Seong, Apu Kapadia, and Donald S Williamson. Defending against microphone-based attacks with personalized noise. *Proceedings on Privacy Enhancing Technologies*, 2021.

[44] Zane Ma, James Austgen, Joshua Mason, Zakir Durumeric, and Michael Bailey. Tracing your roots: exploring the tls trust anchor ecosystem. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 179–194, 2021.

[45] Zane Ma, Aaron Faulkenberry, Thomas Papastergiou, Zakir Durumeric, Michael D Bailey, Angelos D Keromytis, Fabian Monrose, and Manos Antonakakis. Stale tls certificates: investigating precarious third-party access to valid tls keys. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, pages 222–235, 2023.

[46] Zane Ma, Joshua Mason, Sarvar Patel, Manos Antonakakis, Mariana Raykova, Zakir Durumeric, Phillipp Schoppmann, Michael Bailey, Karn Seth, Sascha Fahl, et al. What's in a name? exploring {CA} certificate control. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4383–4400, 2021.

[47] MalShare. Malshare. https://malshare.com/, 2024.

[48] Francois Marier. Enhancing download protection in firefox. https://blog.mozilla.org/security/2016/08/01/enhancing-download-protection-in-firefox/, 2016.

[49] Microsoft. Windows authenticode portable executable signature format. http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fded599bac8184a/Authenticode_PE.docx, 2008.

[50] Microsoft. Winverifytrust function (wintrust.h). https://learn.microsoft.com/en-us/windows/win32/api/wintrust/nf-wintrust-winverifytrust, 2021.

[51] Microsoft. Microsoft defender smartscreen frequently asked questions. https://fb.smartscreen.microsoft.com/smartscreenfaq.aspx, 2024.

[52] Microsoft. Playwright for .net. https://playwright.dev/, 2025.

[53] Abhijit Mohanta, Anoop Saldanha, Abhijit Mohanta, and Anoop Saldanha. Windows internals. *Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware*, pages 123–162, 2020.

[54] Mozilla. Security/safe browsing/chromium implementation overview. https://wiki.mozilla.org/Security/Safe_Browsing/Chromium_Implementation_Overview, 2024.

[55] Harun Oz, Ahmet Aris, Albert Levi, and A Selcuk Uluagac. A survey on ransomware: Evolution, taxonomy, and defense solutions. *ACM Computing Surveys (CSUR)*, 54(11s):1–37, 2022.

[56] Soumajit Pal, Prabaharan Poornachandran, Manu R Krishnan, AU Prem Sankar, and Parvathy Sasikala. Malsign: Threat analysis of signed and implicitly trusted malicious code. In *2017 International Conference on Public Key Infrastructure and its Applications (PKIA)*, pages 23–27. IEEE, 2017.

[57] Mario Polino, Andrea Continella, Sebastiano Mariani, Stefano D'Alessio, Lorenzo Fontana, Fabio Gritti, and Stefano Zanero. Measuring and defeating anti-instrumentation-equipped malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, pages 73–96. Springer, 2017.

[58] The Chromium Projects. Chromium browser source code. https://source.chromium.org/chromium/chromium/src/+/main:chrome/browser/, 2024.

[59] The Chromium Projects. Safe browsing. https://www.chromium.org/developers/design-documents/safebrowsing/, 2024.

[60] Python. Python. https://www.python.org/.

[61] W. Ford VeriSign W. Polk NIST D. Solo Citicorp R. Housley, SPYRUS. Internet x.509 public key infrastructure certificate and crl profile. https://www.ietf.org/rfc/rfc2459.txt, 1999.

[62] Soundarya Ramesh, Ghozali Suhariyanto Hadi, Sihun Yang, Mun Choon Chan, and Jun Han. Ticktock: Detecting microphone status in laptops leveraging electromagnetic leakage of clock signals. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2475–2489, 2022.

[63] Salwa Razaulla, Claude Fachkha, Christine Markarian, Amjad Gawanmeh, Wathiq Mansoor, Benjamin CM Fung, and Chadi Assi. The age of ransomware: A survey on the evolution, taxonomy, and research directions. *IEEE Access*, 11:40698–40723, 2023.

[64] Priyam Kaur Sandhu and Sanjam Singla. Google safe browsing-web security. *IJCSET*, 5(7):283–287, 2015.

[65] Quirin Scheitle, Oliver Gasser, Theodor Nolte, Johanna Amann, Lexi Brent, Georg Carle, Ralph Holz, Thomas C Schmidt, and Matthias Wählisch. The rise of certificate transparency and its implications on the internet ecosystem. In *Proceedings of the Internet Measurement Conference 2018*, pages 343–349, 2018.

[66] Larry Seltzer. Securing your private keys as best practice for code signing certificates. *Viitattu*, 15:2018, 2013.

[67] Trevor Smith, Luke Dickinson, and Kent Seamons. Let's revoke: Scalable global certificate revocation. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[68] Sudhakar and Sushil Kumar. An emerging threat fileless malware: a survey and research challenges. *Cybersecurity*, 3(1):1, 2020.

[69] Virus Samples Team. Github. https://github.com/Virus-Samples/Malware-Sample-Sources, 2024.

[70] Joseph Thio. Fastsigcheck. https://github.com/ThioJoe/FastSigCheck, 2024.

[71] Kurt Thomas, Juan A Elices Crespo, Ryan Rasti, Jean-Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, et al. Investigating commercial {Pay-Per-Install} and the distribution of unwanted software. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 721–739, 2016.

[72] Daniel Uroz and Ricardo J Rodríguez. On challenges in verifying trusted executable files in memory forensics. *Forensic Science International: Digital Investigation*, 32:300917, 2020.

[73] Said Varlioglu, Nelly Elsayed, Zag ElSayed, and Murat Ozer. The dangerous combo: Fileless malware and cryptojacking. *SoutheastCon 2022*, pages 125–132, 2022.

[74] VirusTotal. Virustotal. https://www.virustotal.com/, 2024.

[75] Mike Wood. 'want my autograph?': The use and abuse of digital signatures by malware. 2010.

[76] Mingfu Xue, Chongyan Gu, Weiqiang Liu, Shichao Yu, and Máire O'Neill. Ten years of hardware trojans: a survey from the attacker's perspective. *IET Computers & Digital Techniques*, 14(6):231–246, 2020.

[77] Tianyuan Yu, Hongcheng Xie, Siqi Liu, Xinyu Ma, Xiaohua Jia, and Lixia Zhang. Certrevoke: a certificate revocation framework for named data networking. In *Proceedings of the 9th ACM Conference on Information-Centric Networking*, pages 80–90, 2022.

[78] Chengyuan Zhang, Changqing An, Tao Yu, Zhiyan Zheng, and Jilong Wang. Investigate and improve the certificate revocation in web pki. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–5. IEEE, 2024.

[79] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraş, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 489–502, 2014.