Code Signing Certificate Misissuance in the Wild

Mason Hyman, Julia Steed

Abstract—**In the digital landscape, code signing certificates play a pivotal role in establishing the authenticity and integrity of software applications, ensuring that end-users can trust the source and integrity of the code they execute. However, the increasing prevalence of software supply chain attacks raises significant concerns about the overall security of software distribution. To contribute to this ongoing issue in cybersecurity, this paper investigates code signing certificate misissuance by Certificate Authorities (CAs). In examining the most common misissuances found in the wild and their implications, our research has found that the code signing landscape needs important updates both by CAs and requirement issuing bodies like the Certificate Authority / Browser (CA/B) Forum to provide the trust assurances needed within today's ever-evolving threat landscape.**

Keywords—code-signing, certificate authorities, misissuance, CA/B Forum, security, integrity

I. Introduction

Current software distribution, most commonly through internet downloads, depends on supporting public key infrastructure (PKI) to ensure the integrity and authenticity of data. Composed of thousands of certificate authorities (CAs), the heart of modern PKI revolves around the digital certificate. Because of their application across the internet, digital certificates come in numerous different types and are used for different purposes. Code signing certificates, a type of digital certificate, enable software providers and consumers to verify that their software has not been tampered with or malformed in any way during transmission.

To provide better standardization as well as encourage secure issuance practices, organizations like the Certificate Authority and Browser Forum (referred to as the CA/B forum herein) have been established. The CA/B forum is a voluntary consortium of CAs, internet browser vendors, operating system companies, and other PKI-enabled organizations that issue industry guidelines and requirements related to the issuance and management of X.509v3 certificates [1]. As a part of its mission to increase internet security, the CA/B Forum has published the Baseline Requirements for Code-Signing Certificates (BRfCSC) document that details, among many things, the required profile for code signing certificates.

Because of the global nature of PKI, it is often difficult to fully enforce documents like the BRfCSC without unintended consequences. Portability and compatibility are two foundational staples of the modern internet that often cannot be upheld when constrained by requirements. Despite this fact, popular vendors like Microsoft chose to adopt and enforce the requirements to further improve the security of their products. Accounting for around 90% of the desktop operating system market, Microsoft's choice to embrace the guidelines in late 2016 served to legitimize the baseline both for CAs and consumers.

Still, over seven years after the initial publication of the baseline requirements, code signing certificates are still being misissued due to implementation errors, unclear requirements, and indifference by certificate authorities. Our research attempts to quantify misissuance to better understand the mistakes commonly made when issuing code signing certificates to customers. Additionally, the research attempts to address how CA size impacts misissuance rates. In exploring these questions, we hope to be able to make conclusions about broader PKI security, specifically as it pertains to code signing.

II. Related Work

Our work in this paper has been primarily motivated by Kumar et al.'s research: *Tracking Certificate Misissuance in the Wild* [2]. As the original creators of ZLint, Kumar et al. focused on observations solely in regards to TLS certificates. We have extended their work to code-signing certificate misissuance in this paper in order to broaden the understanding of the greater certificate ecosystem.

Additionally, we were motivated by *A Complete Study of P.K.I (PKI's Known Incidents)*, written by Serrano et al. This paper explores PKI as a whole and introduces the idea of certificate governance. By identifying major stakeholders involved in certificate governance, many of the misissuance cases identified by Kumar et. al. and our own research can be placed into context when discussing their greater impact on web security.

III. Methods

A. Data Collection

The data we collected consisted of four individual datasets consisting of a varying number of code signing certificates. Each dataset was sourced from open source datasets found across the web. In the case of the Sorel dataset, we used specific queries against their large dataset of malware in order to narrow down and clean the total number of samples. Because of the difficulty associated with collecting code signing certificates, which is discussed in more depth in the Discussion section, our datasets were sourced from malware collections. Each malware collection consisted of a number of malicious binaries from which code signing certificates were automatically extracted. It should be noted that we performed no further corresponding analysis into the malware binaries as there appeared to be little correlation between malware type and certificate misissuance.

As shown in Table 1, we divide our certificate datasets based on assumed issuance date. This assumed issuance date was derived from each certificates *notBefore* field that specifies when a code signing certificate becomes valid for use by its holder. Because certificates lack a direct field that tells when they are signed by the CA, we made the assumption that the issuance date was close enough to the date listed in the *notBefore* field in most instances. This is because, most commonly, CAs want to produce certificates that the owner can use immediately.

Further, we chose to only include certificates with a *notBefore* date after the publication of the BRfCSC in our analysis. This choice was made because it would be unfair to

TABLE I. CERTIFICATES IN DATASETS ISSUED AFTER BRFCSC

Dataset:	**Total Certificates**	**Issued After BRfCSC**	**Percent of Total**
Malcert	6,322	0	0%
Symantec	132,485	10,270	7.75%
Sorel	22,559	7,968	35.32%
VirusShare	20,121	1,561	7.76%
All	181,487	19,799	10.91%

[a] Malcert is left out of further discussion because it did not contain applicable certificates.

hold CAs accountable before the requirements were published. Table 1 shows the number of certificates from each dataset that were issued after September 21, 2016, the publication date of the BRfCSC:

B. *Data Processing*

To best analyze large collections of code signing certificates, our research required a fast, modular X.509 certificate linter. We chose ZLint, a tool already written by Kumar et al. to fulfill this role because of its ability to statically analyze certificates by requirement. The ability to add functionality by individual requirement enabled us to add, subtract, and modify lints according to the BRfCSC. Because ZLint was originally created to lint TLS certificates, significant modification was needed for the tool to properly analyze code signing certificates. Our process for modifying the ZLint repository to apply to code-signing certificates was as follows:

- First, we analyzed the currently accepted baselines for code-signing certificates. We created an Excel spreadsheet listing lints that were already present in ZLint and applied both to code signing and TLS certificates, lints that needed partial modification, and lints that were not present. Some lints needed to be updated to correspond with the newest version of the baseline requirements from CA/B forum.

- Next, we assigned lints to specific team members to begin implementation. This was straight forward for lints already present in ZLint but was less straightforward for lints partially present or not present. When creating test cases for lints, we often had to generate custom certificates if the criteria we needed to test was not found in any of the test certificates in the ZLint repository.

- After implementing the lints required to cover the BRfCSC, we were ready to begin analyzing code signing certificates.

```
func (l *subCertDigSigNotSet) CheckApplies(
    c *x509.Certificate) bool {
  return util.IsSubscriberCert(c) &&
    util.IsExtInCert(c, util.KeyUsageOID)
}

func (l *subCertDigSigNotSet) Execute(
    c *x509.Certificate) *LintResult {
  if c.KeyUsage&x509.KeyUsageDigitalSignature != 0 {
    return &LintResult{Status: Pass}
  } else {
    return &LintResult{Status: Error}
  }
}

func init() {
  RegisterLint(&Lint{
    Name:          "e_sub_cert_digital_signature_not_set",
    Description:   "Subscriber Certificate: keyUsage The bit
                    position for digitalSignature MUST
                    be set.",
    Citation:      "BRs: 7.1.2.3.e",
    Source:        MinimumRequirementsForCodeSigningCertificates,
    EffectiveDate: util.MRfCSCEffectiveDate,
    Lint:          &subCertDigSigNotSet{},
  })
}
```

CODE BLOCK 1: EXAMPLE LINT

This process of updating ZLint's functionality to work with code signing certificates in addition to interpreting the baseline requirements, took a large portion of our time and was a major hurdle to overcome. After our updates, ZLint consists of 48 lints with 97% coverage over certificate-related requirements from the BRfCSC. It should be disclosed that there are requirements within the baseline requirements that cannot be verified by static observation of certificates alone. An example of this would be the verification of a Subject's legal name in the *subject:commonName* field as detailed in BRfCSC v3.4 Section 7.1.4.2.2. In this instance, ZLint may verify that the common name field is present, as required, but may not verify that the value is actually the Subject's legal name.

As setup and implemented into ZLint by Kumar et al. lints can be categorized within multiple severity levels. These include: Notice, Warning, and Error which correspond to word mappings between associated requirement documents. In the case of the BRfCSC, the word "should" maps to a Warning severity level, while the word "must" maps to an Error severity level. In our results and discussion, both Warnings and Errors are treated as a misissuance unless stated otherwise. For example, the requirement that THIS EXTENSION MUST BE PRESENT would be classified as an error if not fulfilled. As shown in Code Block 1, severity can be assigned directly in ZLint through conditional return values in the Execute function. In the example, the lint carries an Error severity if the condition that the Digital Signature bit within the Key Usage field is not set.

By dividing lints into functional units that correspond to individual requirements, the programmer is given a greater degree of control over analyzing each certificate. One helpful feature is the ability to set the *EffectiveDate* of each lint. For many of the lints applicable to the BRfCSC this date was set to the publication date of the specific requirement. Because the baselines are continually evolving, with 17 versions being published since the initial document was released, the *EffectiveDate* field enables a greater degree of control when analyzing certificates.

As we began our analysis, we found that ZLint did well to fulfill our needs of a fast and modular linting program. As measured across multiple datasets and runs of the program, ZLint was able to process roughly 250,000 certificates per hour. While this is a slowdown from the 320,000 certificates per hour suggested by Kumar et al, the original authors, our fork chose to output results to an SQLite database. Based on performance metrics collected during execution of the program on large quantities of certificates, it was found that database insertion statements were the most likely culprit for the slowdown. It should be noted that the choice to incorporate a database into the program was not the choice of the authors of this paper, but rather former researchers working on a

TABLE II. MISISSUANCE QUANTIFIED

Dataset	**Misissued**	**Number of Errors**	**Number of Warnings**
Symantec	1,248 (12.15%)	548	913
Sorel	706 (8.86%)	2,040	476
VirusShare	237 (16.40%)	74	182
All	2,192 (11.07%)	2,662	1,571

[a.] Number of Errors and Number of Warnings is total, not by unique certificate.

similar project. Despite this, we did not find the slowdown to be an issue, due in part to our small corpus of certificates.

IV. RESULTS

After running ZLint against the 19,799 code-signing certificates that were issued after September 21st, 2016, we identified that a total of 2,192 certificates were misissued. It should be noted that we included certificates that yielded both errors and warnings in our misissuance count as both contribute to the security and functionality of the individual certificates. Shown in Table 2, our initial finding was that a total of 11.1% of unique certificates were misissued by a certificate authority. This aggregate percentage is much higher than expected, especially when compared to the 3.3% misissuance rate of TLS certificates detected by Kumar et. Al.

As shown in Figure 1, when divided by individual lints, we observed that unsupported signature algorithms were the most common type of misissuance by a margin of over 1,000 occurrences. While the lint name carries an error severity, it should be noted that this is because that is the maximum severity associated with the requirement. In the case of *e_signature_algorithm_not_supported*, the lint may still return a warning result as it aligns with the BRfCSC. We found that in 83% of instances, this specific lint returned a warning instead of an error.

While the signature algorithm field is necessary for CAs to get correct, in many of the instances we examined manually, the lint was triggered by certificates issued near the transition date between RSA-2048 and RSA-3072. As the industry standard for cryptographic protocols changes, the BRfCSC must also adapt. Across numerous versions of the BRfCSC, the required transition date between 2048 bit and 3072 bit RSA changed numerous time. While this likely gave the CAs more flexibility in adopting the change, the required transition date became confused and unclear. We chose to set the effective date within ZLint to the latest date specified in the BRfCSC to prevent misclassifying certificates signed with RSA-2048 as misissued before January 31st, 2017. Still, it is clear from our results and exploration of the certificates themselves, that this date was not a hard transition between the two signature algorithms. This result is discussed in further detail within the Discussion section.

Further, the remainder of the misissuances may be categorized into the following types:

- X.509v3 extension violations: Certificate extensions provide a way for CAs to add additional information and usage restrictions to certificates. With regards to code signing certificates, their importance is paramount to minimizing the vulnerability of the certificate.
- Name form violations: The name form specifies necessary information to identify the holder of the certificate. Example fields include the common name, organization name, locality or state name, and country name.
- Revocation violations: When signed, CAs must include necessary information for validation of the certificate's status. In the case of X.509v3 certificates, the Authority Information Access (AIA) and the Authority Key Identifier extensions contain the HTTP addresses of a CA's Certificate Revocation List (CRL) or OCSP Responder.

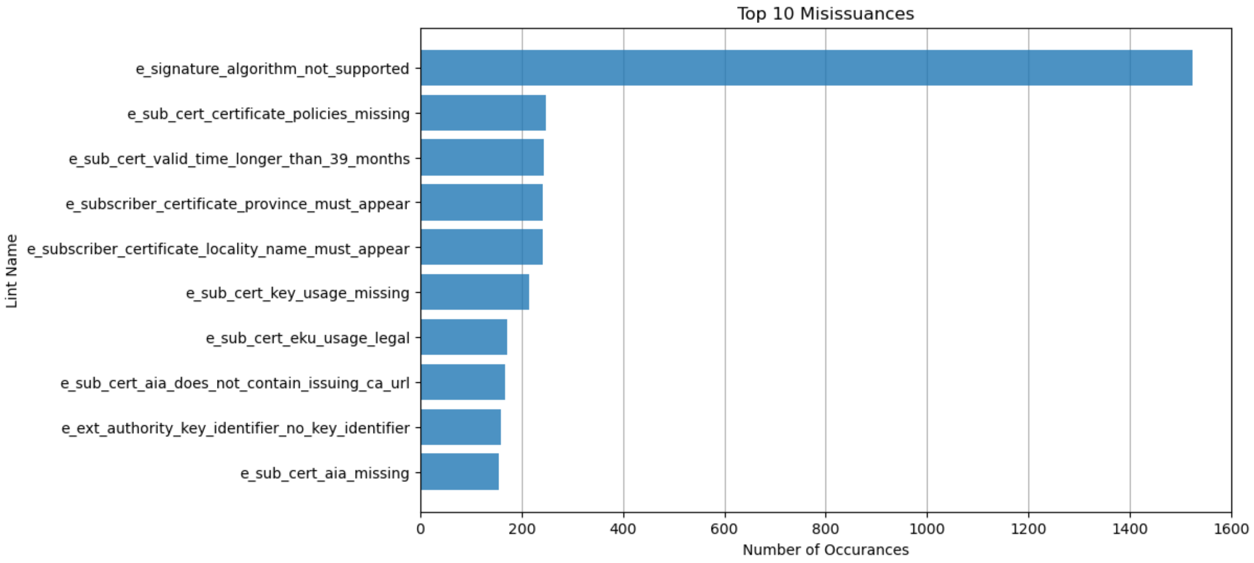

Fig. 1. The top 10 most common misissuances found within code signing certificates.

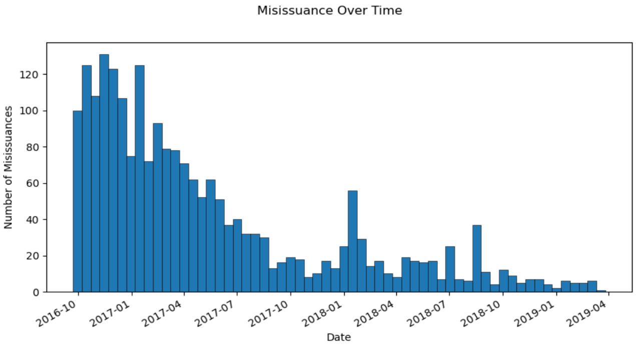

Fig. 2. Misissuance of code signing certificates tracked since the publication of the BRfCSC in September 2019.

While each of these categories do have potential to overlap when examining specific misissuances, they do well to categorize the issue of misissuance as a whole. Each category carries its own concerns as it relates to certificate issuance.

As shown in Fig. 2, misissuance has gone down over time, since the publication of the BRfCSC. While this appears good, a majority of the certificates we collected were issued between 2016-17 which would explain why most misissuances occurred during this time frame. A much greater number of certificates are needed to better explore how misissuance has changed over time. This topic is discussed further within the Discussion section.

To best understand how to address certificate misissuance, it is critical to understand who is responsible for the misissuances themselves. Within ZLint, we captured the issuing certificate authority for each certificate and ranked them by misissuance count. According to Fig. 3, misissuance was most common amongst larger CAs. This is to be expected as larger CAs are issuing a far greater number of certificates than smaller CAs. Our research found that smaller CAs, while not contributing greatly to the total number of misissuances, were far more likely to misissue their select few certificates. While this is the case, a few large certificate authorities like VeriSign Inc. had a high misissuance percentage. This data aligns similarly with the research conducted by Kumar et al. As a whole, large CAs typically misissue smaller percentages of their certificates than smaller CAs. This can be attributed to the fact that larger organizations have greater resources and may contribute directly to the CA/B Forum consortium.

V. Discussion

It is clear from the results of this research that further work needs to be done within the PKI community to provide better security for certificate customers and consumers. Because code signing certificates are responsible for verifying the integrity and authenticity of executable code, they should be treated with great importance. In cases where code signing certificates have been exploited or used maliciously by bad actors, significant damage has been done. Because of this, certificate authorities should devote considerable attention to verifying the correctness of each certificate before issuing it to a customer. Based on the three primary categories of misissuance we derived from our datasets, CAs should be validating the following for every certificate:

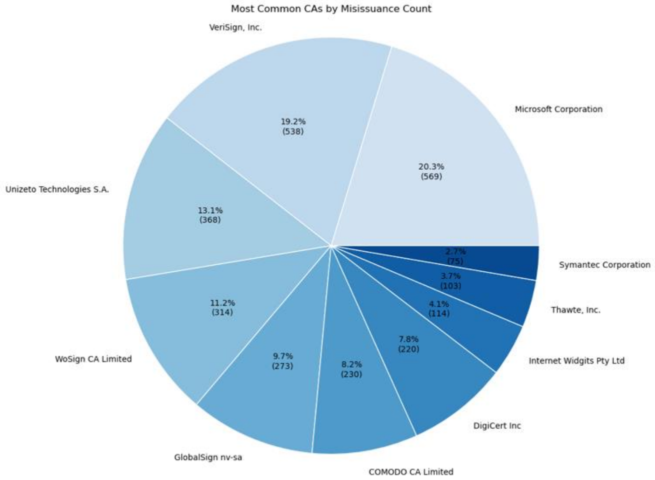

Fig. 3. Top 10 Certificate Authroities that missisue the most.

- All X.509v3 extension fields: These extensions ensure that the certificate is set up to only be used for code signing and may provide important usage restrictions that the customer desires.
- The Name Form of the certificate: The name form contains important information as it relates to the holder of the certificate and the issuing certificate authority.
- Fields that pertain specifically to revocation: Being able to validate revocation is incredibly important to the lifetime of the certificate and those that wish to verify the validity of the certificate. Without this information, the certificate should be trusted only with extreme care.

If these three primary categories are checked according to the baseline requirements prior to issuance, many of the vulnerabilities associated with code signing will be mitigated. When a certificate is misissued according to one of these categories, it opens both the certificate holder and the certificate holder’s customers up to unnecessary liabilities.

To gain better insights into certificate misissuance, our research requires a much greater collection of code signing certificates. As detailed in the Methods section, we used datasets of malicious binaries in order to extract certificates. To get a more holistic picture, we hope to extend this research to certificates attached to benign binaries as well. Unfortunately, there exists a lack of immediate sources for benign binaries and their collection is computationally expensive, specifically in terms of bandwidth and storage. We hope to address this issue in future work such that we may expand knowledge about the code signing ecosystem. Additionally, we hope to examine misissuance as it relates to the difference between Extended Validation (EV) and Non-EV code signing certificates.

VI. GROUP WORK OVERVIEW

To complete this project, work was divided equally amongst both team members. Table III shows individual tasks and who completed them. It should be noted that although one team member is listed per task, each item required direct input from the other team member.

TABLE III. TASK OVERVIEW

Task	**Team Member**
Collect Malcert, VirusShare, and Symantec datasets:	Julia
Collect Sorel dataset:	Mason
Create requirement matrix between existing lints and needed lints:	Julia and Mason
Implement name form lints:	Julia
Implement extension lints:	Mason
Implement misc. lints:	Julia and Mason
Run ZLint on datasets and collect results:	Mason
Perform analysis on ZLint results:	Julia
Write final paper:	Julia and Mason
Create final presentation:	Julia and Mason

VII. REFERENCES

[1] [1] “Cab forum,” CAB Forum, https://cabforum.org/ (accessed Dec. 3, 2023).

[2] D. Kumar et al., “Tracking certificate misissuance in the wild,” 2018 IEEE Symposium on Security and Privacy (SP), 2018. doi:10.1109/sp.2018.00015