Investigating the Health State of X.509 Digital Certificates

Simone Orlando, Alessandro Barenghi, and Gerardo Pelosi

Department of Electronics, Information and Bioengineering - DEIB
Politecnico di Milano, 20133 Milano, Italy
Email: simone.orlando@polimi.it, alessandro.barenghi@polimi.it, gerardo.pelosi@polimi.it

2024 IEEE International Conference on Cyber Security and Resilience (CSR) | 979-8-3503-7536-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSR61664.2024.10679412

***Abstract*—Digital certificates are one of the fundamental components of secure communication protocols where endpoint authentication is desired. However, despite their fundamental role, widespread and widely used libraries approach the certificate validation problem through handcrafted parsers, which are hard to check for correctness. In this work, we investigate the current state of health of the X.509 digital certificate ecosystem, focusing on the ones employed in TLS communications. We perform a comparative analysis of the syntactic and semantic validation capabilities of existing libraries, on a total of 30 million certificates, collected from openly available HTTPS endpoints. Our findings highlight how the current state of health of the X.509 certificates ecosystem has improved in the last 5 years, mainly due to the convergence of the root certification authorities into a small set of highly reliable ones. Despite this, we practically validate the fact that current X.509 managing libraries have significant discrepancies in validation, leading to a large number of certificates being either valid or invalid, depending on the library performing the validation.**

***Index Terms*—Digital Certificates, X.509 format, Parsing**

I. Introduction

Digital certificates play a vital role in securing everyday Internet communications, particularly since their widespread use in protocols such as the Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS). These protocols rely on digital certificates to establish secure bindings between communicating entities and their respective public keys, ensuring endpoint authentication. TLS allows to obtain, transparently, confidential and endpoint-authenticated communications for any application-level protocol that can be transported by TCP alone. Common examples of its use include Hypertext Transfer Protocol Secure (HTTPS) for web browsing, File Transfer Protocol Secure (FTPS) for file transfers, and Simple Mail Transfer Protocol Secure (SMTPS).

Given the widespread need for interoperability in the use of digital certificates, the X.509 standard was soon defined [1], clearly stating the certificate structure and the Distinguished Encoding Rules (DER) format for the encoding of its contents. However, despite a longstanding standard, ensuring syntactic correctness and adherence has proven challenging. Indeed, from a formal standpoint, the language of the valid certificates is *context-sensitive* [2], [3], preventing an efficient and provably effective parser from being automatically generated. The significant complexity of devising a parser for X.509 has led many long-standing libraries in need of handling digital certificates to resort to handcrafted parsers, in turn, leading to a significant risk of mis-validations and inconsistencies among the results provided by them. Such handcrafted parsers are a dual threat to the security of the applications employing such libraries; indeed, mis-parsing a certificate, and deeming it semantically correct afterwards can result in a total loss of the subject-public key binding guarantee it is providing. Furthermore, handcrafted parsers have been affected by a significant amount of common memory management vulnerabilities, which in turn allowed to hijack the control flow of the program embedding them. Classical randomized testing techniques (fuzzing) were proposed to cope with this issue in 2014 by [4], where the authors employed valid, spliced certificates to maximize the code coverage, with respect to fully random inputs. Despite the methodology proposed in [4], the state of effectiveness in the validation of common TLS libraries was still found to be less than satisfactory [2]. In 2018, the authors of [2] took a different approach to the problem, building a LL(*) parser for X.509 certificates, after deriving a formal grammar for the language, excluding intrinsically ambiguous conditions. The results of the (purely syntactical) analysis made by the parser were compared to the ones of the full (i.e., syntactic and semantic) validation of X.509 certificates performed by seven widespread TLS managing libraries. The authors reported that essentially one certificate in five among the ones retrieved via a full IPv4 address space scan did not pass syntactic validation; despite this, a relevant portion of them was accepted as semantically valid by most cryptographic libraries.

Contribution. In this study, we aim to assess the current state of soundness of digital certificates, evaluating their syntactic correctness, and performing cross-validation among popular TLS libraries. Our aim is to investigate whether the fragility of the ecosystem reported in [2] is still present, calling for further action, or the outlook is now a more comforting one. To this end, we employ a fresh dataset of 30 million certificates collected on April 24, 2024, and perform a cross-comparison between syntactic evaluation and semantic validation by eight widespread TLS libraries. We report significant differences in the number of syntactic errors among certificates issued by well-known and lesser-known Certificate Authorities (CAs). The former exhibit minimal errors, involving only

a few hundred certificates. In contrast, the latter, especially CAs issuing certificates for internal use rather than public consumption, show a notable increase in errors, totaling up to 4.25 million certificates. Our analysis not only identifies syntactic errors but also categorizes them based on severity to understand potential security implications. Among the most prevalent errors, we identify five critical errors, with the most common affecting 3.58 million certificates. Furthermore, our comparison of TLS libraries reveals substantial variability in the number of rejected certificates, highlighting inconsistencies across implementations.

Structure of the work. The rest of this paper is structured as follows: Section II provides background information on the X.509 standard and parser generators, while Section III outlines our methodology, including tools and datasets. Section IV presents our findings, including the number of invalid certificates, their issuing CAs, the detected syntactic errors, and parsers comparison. Finally, Section V offers conclusions.

II. Background

In this section, we outline the key features established by the X.509 standard for defining the syntactic structure of digital certificates, along with an introduction to parser generators and their role in automatically generating language parsers from grammar descriptions.

A. X.509 Standard

The X.509 format stands as a widely embraced standard in the realm of Public Key Infrastructure (PKI) for managing digital certificates and public key-based encryption. First introduced in 1988 as part of the X.500 standard by the International Telecommunication Union (ITU) [5] and further described in a series of Request for Comments (RFCs) memoranda [6]–[11], X.509 provides a scalable and efficient method for verifying the authenticity of bindings between entities, such as individuals or organizations, and their corresponding public keys. The specification employs Abstract Syntax Notation 1 (ASN.1) meta-syntax [12] and consists of three primary Abstract Data Types (ADTs): `TBSCertificate`, `SignatureAlgorithm`, and `SignatureValue`.

The `TBSCertificate` is the field signed by the Certificate Authority (CA) and encompasses various details such as the certificate version, serial number, validity period, issuer's name, owner's name, public key, signing algorithm used by the issuer, and optional fields, including unique identifiers and extensions. The signature produced by the CA is stored in the `SignatureValue` structure, while the algorithm employed for generating the signature is specified in the `SignatureAlgorithm` field.

Figure 1 illustrates a concise depiction of the X.509 digital certificate structure, showcasing its main ASN.1 ADTs.

In real-world applications, X.509 certificates are predominantly serialized utilizing the Distinguished Encoding Rules (DER) as outlined in X.690 standard [13]. The DER encoding strategy revolves around encoding ASN.1 ADTs as a sequence of bytes through a triplet structure of the form $\langle t, \ell, v \rangle$, where t signifies the tag indicating the data type, ℓ represents the length of the data, and v indicates the actual data content. The v field may, in turn, nest another ADT encoded similarly using the triplet encoding. Parsing such a binary structure proves to be a complex task, necessitating careful analysis of the length field to correctly interpret the data field. Since the internal grammar of a DER-encoded certificate is context-sensitive, developing an accurate parser for this grammar is not straightforward. This intrinsic complexity is likely the reason why modern libraries analyze and manage digital certificates through fully handcrafted implementations of parsers.

```
Certificate ::= SEQUENCE {
tbsCert            TBSCertificate,
signatureAlgorithm AlgorithmIdentifier,
signatureValue     BIT STRING }

TBSCertificate ::= SEQUENCE {
version [0] EXPLICIT Version DEFAULT v1,
serialNumber         CertSerialNumber,
signature            AlgorithmIdentifier,
issuer               Name,
validity             Validity,
subject              Name,
subjectPublicKeyInfo SubjectPubKeyInfo,
issuerUniqueID  [1] IMPLICIT UniqueId   OPTIONAL,
subjectUniqueID [2] IMPLICIT UniqueId   OPTIONAL,
extensions      [3] EXPLICIT Extensions OPTIONAL }

AlgorithmIdentifier ::= SEQUENCE {
algorithm  OBJECT IDENTIFIER,
parameters ANY DEFINED BY AlgorithmP OPTIONAL}

SubjectPublicKeyInfo  ::=  SEQUENCE  {
algorithm         AlgorithmIdentifier,
subjectPublicKey  BIT STRING }
```

Fig. 1. Part of the ASN.1 description of the X.509 standard.

B. Parser Generators

The task of parsing a language involves analyzing its structure based on a formal grammar representation, which defines the rules for determining which sequences of symbols are valid within the language. These rules are commonly expressed using notations such as the Backus-Naur Form (BNF) or the Extended Backus-Naur Form (EBNF), providing a clear and structured way to describe the language's syntax. Parser generators are tools designed to simplify and automate the creation of parsers. A parser generator takes a formal grammar description as input and produces the source code for a functional parser in a specific target programming language. The main advantage of using parser generators is that they significantly reduce the likelihood of human errors in parser development, ensuring greater reliability and consistency in data interpretation. However, parser generators are not without limitations. Some grammars may have intrinsic complexities that make it difficult, if not impossible, to check all the prescribed constraints exclusively with a parser generator. This is particularly the case with the grammar standardized to describe the X.509 format, where the looseness in the specification of some fields or the ambiguity of the description of some portions of a certificate may lead to the undecidability

TABLE I
CLASSIFICATION OF SYNTACTIC ERRORS ACCORDING TO THEIR IMPACT ON SECURITY.

Security Critical Errors	Non Security Critical Errors
KEY USAGE CONSTRAINT	MISSING AUTHORITY KEY IDENTIFIER
WRONG ALGORITHM OID	EMPTY RELATIVE DISTINGUISHED NAME
UNEXPECTED KEY CERT SIGN BIT	PATH LENGTH CONSTRAINT IN NOT CRITICAL BASIC CONSTRAINTS
WRONG STRING TYPE	DISTINGUISHED NAME STRING ERROR
EXTENSION IN OLD VERSION CERTIFICATE	MISSING SUBJECT KEY IDENTIFIER
MISSING KEY CERT SIGN BIT	PRINTABLE STRING REGEX ERROR
UNEXPECTED NULL ALGORITHM IDENTIFIER PARAMETERS	WRONG POLICY OID
LEXING ERROR	NOT CRITICAL BASIC CONSTRAINTS EXTENSION
DUPLICATED EXTENSION	EMPTY PRINTABLE STRING

of parsing. Barenghi et al. [2] proposed and demonstrated the effectiveness of a systematically generated parser for the syntactic checking of X.509 certificates. They exploit a relaxed version of the X.509 grammar and the `ANTLR` parser generator [14], with `C` as the target language. They supplemented the generated parser with handwritten semantic rules to manage the small portion of the standard that could not be handled automatically, ensuring an accurate process of all X.509 certificates.

III. CERTIFICATE SURVEY AND VALIDATION METHODOLOGY

To access a comprehensive dataset of digital certificates for analysis, we utilized the Censys platform [15]. Censys provided us with a snapshot of 30.15 million certificates collected through Internet-wide scans and synchronization with Certificate Transparency (CT) servers on April 24, 2024. The dataset consists of $2,485$ serialized `Avro` files containing raw certificates, individual certificate fields, and additional metadata, totalling 47 GB in size. To facilitate easier querying and analysis, the information within these files we extracted and exported them into a MySQL database.

A. Certificate Validation

Certificate validation involves two main aspects: syntactic and semantic validation. Syntactic validation verifies the compliance of a certificate to the X.509 format specifications. Semantic validation focuses on interpreting the information contained in the fields of the certificate, including verifying the issuing CA, checking the certificate's validity period, and confirming the association between the certificate's public key and the subject's identity. To assess the correctness of the collected certificates, we employed the parser developed by Barenghi et al. [2]. Additionally, we wrote minimal clients for each of the following, widely used, TLS libraries: OpenSSL (v3.0.11) [16], which is employed in both the Apache and NGINX web server SSL/TLS modules; BoringSSL (commit: b6bca9c, 2024-04-04) [17], the fork of OpenSSL maintained by Google and used as the default TLS library in Chrome, Chromium, and Android system binaries; GNUTLS (v3.7.9) [18], the Free Software Foundation TLS library employed in the GNOME desktop environment facilities and the Common Unix Printing System (CUPS); Network Security Services (NSS) (v3.90.2) [19], the TLS implementation of the Mozilla Foundation employed in Firefox; WolfSSL (v5.7.0) [20], a lightweight client and server library targeted for embedded, RTOS, and resource-constrained environments; Mbed TLS (v3.6.0 LTS) [21], used among others by ARM in its ecosystem of IoT devices; LibreSSL (v3.9.2) [22], the fork of OpenSSL maintained by OpenBSD and default on their operating system; and BouncyCastle (v1.78.1) [23], the default security provider in Android apps.

All TLS libraries analyzed in this study do not separate syntactic validation from semantic validation and require the complete *chain of trust* (i.e., the sequence of issued certificates from the leaf to the root-CA one) for each certificate to perform a comprehensive analysis. Upon inspecting the dataset obtained from Censys, we observed that for some certificates, the field specifying the chain of trust was missing. Consequently, we divided the initial dataset into two subsets: one containing $21,302,799$ certificates with complete chains and another containing $8,855,696$ certificates without chains. The former underwent analysis employing all mentioned TLS libraries and the Barenghi et al. parser [2], while the latter underwent analysis employing solely the Barenghi et al. parser, as it does not require the entire chain to work, but performs syntactic analysis on individual certificates.

Additionally, we noticed that the Censys database contained *pre-certificates*, a special type of certificate used in the Certificate Transparency (CT) system [24]. Certificate Authorities submit pre-certificates to CT logs to obtain the log's signed certificate timestamp (SCT), a promise by a CT log to publicly record a certificate, that needs to be included in the certificate by the CA. The pre-certificates contain the same information as a final certificate, but with a "poison" extension (OID: 1.3.6.1.4.1.11129.2.4.3) that is marked as critical, which forces TLS libraries to reject them. Since their rejection corresponds to correct behaviour by the libraries, we did not consider them as erroneous.

TABLE II
REJECTIONS OF PAIRWISE INTERSECTIONS AMONG PARSERS, EXCLUDING ERRORS RELATED TO EXPIRATION OR PRECERTIFICATES POISONED EXTENSION. THE TOTAL NUMBER OF CERTIFICATES REJECTED BY AT LEAST ONE PARSER IS 4,390,378.

	Barenghi et al. [2]	**OpenSSL**	**BoringSSL**	**GNUTLS**	**NSS**	**WolfSSL**	**mbedTLS**	**LibreSSL**	**Bouncy Castle**
Barenghi et al. [2]	384	0	0	0	384	0	0	0	0
OpenSSL		3,756,563	3,755,696	1,872,051	9,848	67,085	3,756,428	1,866,250	57,314
BoringSSL			4,050,847	1,871,680	146,611	358,499	3,755,690	1,865,513	348,710
GNUTLS				1,872,141	9,816	9,792	1,872,031	1,865,861	110
NSS					158,622	149,387	9,850	9,876	139,588
WolfSSL						364,501	67,144	13,085	354,650
mbedTLS							4,007,717	1,866,313	57,293
LibreSSL								1,944,407	5,941
Bouncy Castle									354,760

B. Differential Analysis Strategy

One challenge in the validation process we followed is that libraries provide a single response for the validation of an entire certificate chain. This could lead to misinterpretations, where a leaf certificate without errors is erroneously considered invalid due to errors in another certificate along the chain of trust. To address this issue, we employed a differential analysis strategy. In case of an error detected in the chain, we revalidated the chain without the leaf certificate using the same library that reported the error. Thus, a leaf certificate is deemed invalid only if the error reported for the chain ending with it differs from the error reported for the chain ending with the certificate of the CA that issued it. If errors reported for the certificate at hand and the one of its CA during the chain of trust validation process match, the certificate is considered valid. The only risk associated with this approach is that a leaf certificate may be affected by the same error reported when analyzing the certificate of its issuing CA and then deemed valid. However, this is not expected to occur.

C. Classification of Syntactic Errors

Among the possible syntactic errors detectable within the certificates, not all have the same severity and impact on security. In Table I we classified the most common errors into two categories: *security-critical* errors that can potentially lead to exploitable security vulnerabilities, and *non-security-critical* errors that do not pose an immediate security threat. In the following, we delve into the potentially critical errors and provide an outline of how these could be exploited to compromise the security of a system.

A KEY USAGE CONSTRAINT error occurs when there are bits in the Key Usage extension of the X.509 document that are not permitted by the public key algorithm used in the certificate. This issue can cause serious vulnerabilities because the application might use the public key for operations that are not intended to be executed with that key (e.g., encryption vs. signature verification).

A WRONG ALGORITHM OID error happens when an unexpected Object Identifier (OID) is detected in the certificate's algorithm identifier field. This error arises because the parser, as pointed out in [2], should only accept OIDs specified in the standard. Accepting a generic cryptographic primitive as an algorithm may compromise the authenticity of the communication, as the expected cryptographic guarantees may not be provided.

An UNEXPECTED KEY CERT SIGN BIT error occurs when the parser encounters a Basic Constraint extension that is either not critical or where the CA flag is absent or `FALSE`. This is a serious security problem that can be exploited in scenarios similar to [25] to perform powerful impersonation attacks.

A WRONG STRING TYPE error occurs when an unexpected ASN.1 String type is found instead of the valid one. Vulnerabilities related to this error depend on how libraries handle this value since different encodings can lead to different interpretations of the same bytes.

An EXTENSION IN OLD VERSION CERTIFICATE error occurs when extensions are detected in a version 1 or 2 certificate, as these are only allowed in version 3. The security issue here is that an implementation may consider (e.g., for efficiency reasons) only a portion of the certificate to validate its signature, allowing an attacker to add an arbitrary number of extensions with content of their choice.

A MISSING KEY CERT SIGN BIT error occurs when a Basic Constraints extension with a Path Length Constraint has been parsed. This issue could increase the attack surface if there is a faulty check during the validation algorithm of this bit to use the certificate as an intermediate one. A library might correctly check the CA flag and still skip the Key Cert Sign verification, allowing an attacker to use the certificate to sign other certificates even if the Key Cert Sign bit is cleared.

An UNEXPECTED NULL ALGORITHM IDENTIFIER PARAMETERS error is a syntactic issue that occurs when a NULL ASN.1 object is encountered in the Algorithm Identifier field. There could be security implications depending on the application's behaviour in retrieving parameters required for cryptographic purposes.

A LEXING ERROR is raised either because an invalid ASN.1 tag is used or because the length field of a primitive field is incorrect. This can raise security issues when libraries, instead of detecting the ASN.1 encoding error, try to interpret

TABLE III
LIST OF THE MOST PREVALENT ROOT CAS AND THEIR ASSOCIATED INTERMEDIATE CAS BASED ON THE DATASET OF 21.30 MILLION CERTIFICATES WITH RECONSTRUCTED CHAINS INCLUDING THE NUMBER OF CERTIFICATES ISSUED BY EACH INTERMEDIATE CA, AND THE NUMBER OF CERTIFICATES DEEMED VALID BY ALL TESTED PARSERS

Root CA	Intermediate CA	Issued Certs	Valid Certs
ISRG Root X1	Let's Encrypt R3	13,535,997	8,093,757
GTS Root R1	GTS CA 1P5	1,730,258	607,145
	GTS CA 1D4	352,616	140,257
ISRG Root X2	Let's Encrypt E1	1,189,006	815,420
Go Daddy Root G2	Go Daddy Secure G2	753,714	593,086
Amazon Root CA 1	Amazon RSA 2048 M02	471,026	27,492
	Amazon RSA 2048 M03	415,098	21,385
COMODO RSA CA	cPanel CA	354,650	0
USERTrust ECC CA	ZeroSSL ECC DSS CA	348,583	164,470
DigiCert Global Root G2	Encryption Everywhere G2	247,720	120,395

these fields as if they were correctly encoded, leading to interpretation errors that can hide security attacks [3].

Finally, a DUPLICATED EXTENSION error occurs when the uniqueness constraint on the extensions present in the certificate is violated. Possible vulnerabilities can arise based on how libraries handle the validation of these multiple instances.

IV. EXPERIMENTAL RESULTS

In this section, we present an analysis of the data gathered throughout our study. We executed the parser in [2] and all client implementations for the TLS libraries on a Debian 6.1.76-1 (2024-02-01) system, equipped with dual AMD EPYC 7551 32-Core CPU and 512 GiB of DDR-4 DRAM.

A. Dataset with complete chains

The results obtained for the 21 million certificates, for which the Censys dataset provided the metadata to reconstruct the entire chain of trust, are shown in Table III. This table highlights how the CA landscape is predominantly dominated by four main entities: Let's Encrypt, Google, GoDaddy, and Amazon. These four alone account for 86.60% of the 21 million certificates in the dataset, with Let's Encrypt leading the pack by signing 69.12% of the certificates. In addition to the number of certificates issued, the table also shows the number of certificates considered valid both by the parser in Barenghi et al. [2] and by all the considered TLS libraries. Among these, the parser of Barenghi et al. detects only 384 certificates with non-critical syntactic errors, of which 383 are VISIBLE STRING REGEX ERRORS and 1 is a DISTINGUISHED NAME STRING ERROR. All these certificates are issued by the same entity, the Belgian certificate authority Citizen CA. This is consistent with the fact that certificates marked as invalid by TLS libraries are due to semantic errors such as USES INSECURE ALGORITHM and INVALID SIGNATURE. This indicates that the syntactic correctness of certificates signed by the most widely used and well-known CAs is solid.

Table II presents a pairwise comparison of the parsers, where each value indicates the number of certificates considered invalid by both parsers. Errors related to certificate expiration or the presence of the pre-certificates poison extension (see Section III) are not counted, as they are irrelevant for comparing the parsers' ability to reject truly invalid certificates and for assessing the current syntactic condition of digital certificates in circulation. This comparison reveals a significant discrepancy in validation across different parsers, highlighting the challenges in conforming to a standard as complex and ambiguous as X.509. This results in current digital certificate validation libraries exhibiting vastly different behaviours when it comes to rejecting a certificate. For example, GNUTLS and Bouncy Castle agree on rejecting only 110 certificates, while OpenSSL and NSS agree on rejecting just 9,848 certificates. Moreover, our analysis also revealed that the number of certificates rejected by at least one parser is 4,390,378. This figure highlights the extent of behavioural diversity among the parsers, especially when compared to the individual rejection rates of the single libraries or the pairwise intersections. The differences observed between these values are of several orders of magnitude, underscoring the complexity and variability in certificate interpretation across different parsers.

B. Dataset with only leaf certificates

A completely different situation arises concerning the syntactic correctness of the 8.85 million certificates for which Censys did not provide metadata regarding their chain of trust. Table IV lists the most common CAs that signed these certificates, along with the number of certificates deemed syntactically valid by the parser of Barenghi et al. [2].

In this dataset, 48% of the certificates contain syntactic errors. Fortinet stands out, issuing 37.29% of the certificates and accounting for 77.05% of the certificates with errors. The stark difference from the previous scenario might be attributed to the presence of certificates issued by lesser-known and less widespread CAs. These CAs might have less reliable certificate generation systems, possibly because some of these CAs issue certificates exclusively for their devices or ecosystems. Consequently, they might be less incentivized to maintain error-free certificate generation systems compared to CAs that issue certificates to the general public.

Table V presents the types of syntactic errors detected, along with the count of affected certificates and a label indicating whether the error is classified as critical (i.e., can represent an exploitable security vulnerability). Notably, among the most common errors there are four classified as critical, with the Key Usage Constraint error being the most prevalent overall, affecting 40.45% of the certificates in the dataset.

V. CONCLUSION

In this study, we focused on the evaluation of the syntactic correctness of X.509 certificates currently in circulation. Our analysis revealed a considerable contrast between certificates produced by well-known, widely-used CAs and those issued by lesser-known CAs or CAs, which generate certificates

TABLE IV
LIST OF THE MOST PREVALENT INTERMEDIATE CAS BASED ON THE DATASET OF 8.85 MILLION CERTIFICATES WITHOUT CHAINS, INCLUDING THE NUMBER OF CERTIFICATES ISSUED, AND THE NUMBER OF CERTIFICATES DEEMED VALID BY BARENGHI ET AL. PARSER [2].

Intermediate CA	Issued Certs	Valid Certs
Fortinet	3,302,448	26,480
GTS Test WR3	1,506,978	1,506,978
Let's Encrypt E1 (STAGING)	932,192	932,192
Let's Encrypt R3 (STAGING)	231,824	231,824
Empty	218,047	0
Traefik Default Cert	197,805	1,534
Kubernetes	89,777	79,349
Freebox Intermediate CA	78,276	0
HDWA Subordinate CA	67,266	0
Kubernetes IC Fake Certificate	55,859	55,856

TABLE V
ERRORS DETECTED BY BARENGHI ET AL. PARSER [2] ON THE DATASET OF CERTIFICATES WITHOUT CHAINS, ALONG WITH THEIR RESPECTIVE COUNTS AND CRITICALITY CLASSIFICATION.

Error	Count	Critical
KEY USAGE CONSTRAINT	3,582,041	yes
MISSING AKI KEY ID	287,762	no
EMPTY RDNS	217,942	no
PATHLEN NO BC CRITICAL	93,565	no
X509 DNAME	14,629	no
GENERIC ERROR	14,593	no
MISSING SKI	14,345	no
PRINT STRING REGEXP	10,101	no
POLICY WRONG OID	3,464	no
WRONG ALG ID OID	3,199	yes
UNEXPECTED CERT SIGN BIT	1,546	yes
WRONG STRING TYPE	1,473	yes
Others	6,915	-

exclusively for their own devices and ecosystems. Certificates from prominent CAs demonstrated a high degree of syntactic correctness, while those from less established CAs exhibited a higher incidence of errors. We compared the performance of various widely used TLS libraries in certificate validation and discovered significant differences in their rejection rates. This variability highlights how the X.509 standard leaves too much room for interpretation, leading to the development of handcrafted parsers that behave inconsistently when validating such a critical component of Internet security. A future deeper investigation about the usage of the opportunities allowed by context-sensitiveness of the X.509 standard grammar may check to which extent they can be actually found in the set of deployed certificates. This would give more quantitative reasons to restrict or not the flexibility of the current X.509 standard grammar.

ACKNOWLEDGMENT

This work was supported in part by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

REFERENCES

[1] International Telecommunication Union, "Recommendation ITU-T X.509: Open Systems Interconnection," https://www.itu.int/rec/T-REC-X.509-201210-I, 2012.
[2] A. Barenghi, N. Mainardi, and G. Pelosi, "Systematic parsing of X.509: eradicating security issues with a parse tree," *J. Comput. Secur.*, vol. 26, no. 6, pp. 817–849, 2018. [Online]. Available: https://doi.org/10.3233/JCS-171110
[3] D. Kaminsky, M. L. Patterson, and L. Sassaman, "PKI layer cake: New collision attacks against the global X.509 infrastructure," in *FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010*, ser. LNCS, vol. 6052. Springer, 2010, pp. 289–303. [Online]. Available: https://doi.org/10.1007/978-3-642-14577-3_22
[4] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations," in *S&P 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 114–129. [Online]. Available: https://doi.org/10.1109/SP.2014.15
[5] International Telecommunication Union, "Recommendation ITU-T X.500: Open Systems Interconnection," https://www.itu.int/rec/T-REC-X.500-198811-S, 1988.
[6] T. Polk, R. Housley, and L. Bassham, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3279, 2002.
[7] R. Housley, B. Kaliski, and J. Schaad, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL)," RFC 4055, 2005.
[8] S. Leontiev and D. Shefanovski, "Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile," RFC 4491, 2006.
[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and T. Polk, "Internet X.509 Public Key Infrastructure Certificate and CRL," RFC 5280, 2008.
[10] Q. Dang, S. Santesson, K. M. Moriarty, D. R. L. Brown, and T. Polk, "Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA," RFC 5758, 2010.
[11] S. Turner, D. R. L. Brown, K. Yiu, R. Housley, and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information," RFC 5480, 2009.
[12] International Telecommunication Union, "Recommendation ITU-T X.680: Information Tech." https://www.itu.int/rec/T-REC-X.680, 2015.
[13] ——, "Recommendation ITU-T X.690: Information technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)," https://www.itu.int/rec/T-REC-X.690, 2015.
[14] T. Parr and K. Fisher, "LL(*): the foundation of the ANTLR parser generator," in *PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, 2011. [Online]. Available: http://doi.acm.org/10.1145/1993498.1993548
[15] Z. Durumeric, D. Adrian, A. Mirian, M. D. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning," in *CCS 2015, Denver, CO, USA, October 12-16, 2015*. ACM, 2015, pp. 542–553. [Online]. Available: https://doi.org/10.1145/2810103.2813703
[16] OpenSSL Foundation, "OpenSSL: Cryptography and SSL/TLS Toolkit," https://www.openssl.org/, 2024.
[17] Google Inc., "BoringSSL," https://boringssl.googlesource.com/boringssl/, 2024.
[18] Free Software Foundation, Inc., "The GnuTLS Transport Layer Security Library," http://www.gnutls.org/, 2024.
[19] Mozilla Foundation, "Network Security Services," https://wiki.mozilla.org/NSS, 2024.
[20] Todd Ouska, "wolfSSL Embedded SSL/TLS Library," https://www.wolfssl.com/, 2024.
[21] TrustedFirmware, "Mbed TLS," https://www.trustedfirmware.org/projects/mbed-tls/, 2024.
[22] The OpenBSD Project, "LibreSSL," https://www.libressl.org//, 2024.
[23] Legion of the Bouncy Castle, "Bouncy Castle Java Cryptography APIs," https://www.bouncycastle.org/java.html, 2024.
[24] B. Laurie, E. Messeri, and R. Stradling, "RFC 9162: Certificate Transparency Version 2.0," USA, 2021.
[25] A. Langley and D. Benjamin, "OpenSSL Security Advisory - Alternative Chains Certificate Forgery," http://openssl.org/news/secadv/20150709.txt, 2015.