Analyzing Compliance and Complications of Integrating Internationalized X.509 Certificates

Mingming Zhang*
Zhongguancun Laboratory
Beijing, China
zhangmm@mail.zgclab.edu.cn

Jinfeng Guo*
Nankai University
Tianjin, China
g20000217@gmail.com

Yiming Zhang
Tsinghua University
Beijing, China
zhangyiming@tsinghua.edu.cn

Shenglin Zhang
Nankai University
Tianjin, China
zhangsl@nankai.edu.cn

Baojun Liu
Tsinghua University
Beijing, China
lbj@tsinghua.edu.cn

Hanqing Zhao
Tsinghua University
Beijing, China
zhaohq23@mails.tsinghua.edu.cn

Xiang Li
Nankai University
Tianjin, China
lixiang@nankai.edu.cn

Haixin Duan
Tsinghua University
Beijing, China
Quancheng Laboratory
Jinan, China
duanhx@tsinghua.edu.cn

Abstract

The global PKI supports the issuance of *Unicerts*, which are X.509 certificates that integrate internationalized content such as IDNs and multilingual text. This integration introduces complexity in Unicert issuance and usage. Past incidents showed that poor Unicode handling can cause security risks, including spoofing and remote code execution, yet threats specific to PKI and Unicerts remain underexplored. This paper presents the first large-scale study of Unicerts, examining both issuance and parsing compliance. By analyzing 34.8 million Unicerts from CT logs and 9 mainstream TLS libraries, we found the PKI ecosystem struggles with adopting Unicode. On the issuing side, 373 issuers produced 249.3K (0.72%) noncompliant Unicerts due to weak validation on character ranges, normalization, and formatting, of which 65.3% arise from publicly trusted CAs. These issues arise from overly complex standard requirements. On the parsing side, TLS libraries like GnuTLS and PyOpenSSL exhibited issues in decoding and handling special characters, such as incompatible decoding and improper escaping, which could lead to incorrect entity extraction or subfield forgery. We further empirically identified threat surfaces, including user spoofing, CT monitor misleading, and traffic obfuscation. Finally, we analyzed root causes and proposed recommendations to enhance Unicert compliance in the global PKI ecosystem.

CCS Concepts

• **Security and privacy → Security protocols**; • **Networks →** Network measurement.

*Both authors contributed equally to this research.

Keywords

PKI; Measurement; X.509 Certificates; Universal Acceptance

ACM Reference Format:
Mingming Zhang, Jinfeng Guo, Yiming Zhang, Shenglin Zhang, Baojun Liu, Hanqing Zhao, Xiang Li, and Haixin Duan. 2025. Analyzing Compliance and Complications of Integrating Internationalized X.509 Certificates. In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25), October 28–31, 2025, Madison, WI, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3730567.3764483

1 Introduction

Public Key Infrastructure (PKI) is the security foundation of the Internet, employing X.509 certificates to bind identities (e.g., hostnames, organizations) to cryptographic keys. To facilitate a truly multilingual Internet, Universal Acceptance (UA)[1] initiatives promote the adoption of internationalized network identifiers, such as domain names (IDNs) and email addresses, across diverse languages and scripts [44, 97]. While X.509 certificates already support IDNs and most Unicode characters [17], the UA readiness of PKI has remained largely unexamined.

Motivation. Correct processing of identity fields in X.509 certificates is essential. Ensuring its compliance has been a top priority over the years [9, 34, 94], as standard violations can lead to distrust of Certificate Authorities (CAs) by browsers. However, incorporating Unicode in certificate fields complicates the issuance, parsing, and validation process, potentially causing usability and security issues. This complexity also affects scenarios requiring certificate storing and searching, such as traffic analysis, threat detection, and Certificate Transparency (CT) monitoring. Real-world incidents show that improper use of Unicode in certificates can lead to spoofing threats [20, 59, 62, 75], invalid certificate logs [5, 68], and even remote code execution risks [74]. Moreover, certificates with illegal formats or characters can trigger crashes or bypass certificate

[1]Universal Acceptance (UA) is a fundamental requirement for a multilingual and digitally inclusive Internet.

validation [74, 83]. This highlights the need to analyze internationalized certificates (*Unicerts*) for issuance compliance, parsing accuracy, and potential impacts to assess PKI's UA readiness.

Research gaps. On the issuance side, previous work has identified various noncompliance issues [24, 53, 81, 82] and proposed several certificate linters [6, 32, 80, 88, 96], but primarily focuses on regular field value compliance and overlooks Unicode-specific concerns. On the validation side, studies assume successful certificate parsing and focus on the subsequent validation processes [7, 14, 22, 85], with limited analysis of the parsing mechanisms. Although some noticed syntax errors [4, 13, 22, 23] in certificates, they still lacked in-depth analysis of handling Unicode-related fields.

Research questions. This study aims to assess the current UA-readiness in PKI systems and provide recommendations to the security community. Our research questions include: (RQ1) Have CAs issued Unicerts in compliance with standard requirements? (RQ2) Do TLS implementations parse Unicerts according to normative constraints? (RQ3) What are the impacts of noncompliant issuance and parsing flaws?

Our study. We conducted the first large-scale measurement and empirical study of Unicerts. For RQ1, we employed RFCGPT [89, 90], an established custom GPT pretrained on ~2K RFCs, to extract Unicode-related constraints for certificates. It helped navigate interdependent and evolving standards, enabling us to generate 95 constraint rules (50 of which have not been covered in existing studies). We developed a certificate linter equipped with these constraints, and applied it to 34.8 million Unicerts from a CT dataset containing over 70 billion certificates, to quantify real-world issuance compliance. For RQ2, we examined TLS implementations to uncover Unicert parsing anomalies. We constructed test Unicerts with various Unicode blocks (e.g., C0 Controls) and encoding types (e.g., PrintableString), to check APIs in 9 mainstream TLS libraries. For RQ3, we empirically explored potential threats of malformed Unicerts, focusing on scenarios including visualization, CT monitoring, and traffic analysis.

Major findings. Achieving an internationalized PKI remains challenging in issuance and parsing:

For RQ1, we observed that PKI systems have made significant progress in supporting Unicerts (Section 4). A total of 698 CA organizations have issued 34.8 million Unicerts, with over 97.6% from the top 10 CAs (e.g., Let's Encrypt and Sectigo). However, due to the complexity of attribute constraints in X.509 certificates, 373 issuers exhibited three types of noncompliance: improper character checks (43K certs), lack of value normalization (3 certs), and invalid format or structure validation (206K certs). Although the CA/B Baseline Requirements (BRs) regard some of these issues as technically valid, they nonetheless pose real security risks. While the overall impact appears low and we found no evidence of real-world exploitation, these issues can enable specialized attacks, as discussed in Section 4.3.1 and Section 4.4. Moreover, both globally prominent CAs (e.g., DigiCert, Let's Encrypt) and regional ones (e.g., Czech Post, DOMENY.PL) were involved, revealing that such practices are widespread. In particular, 65.3% of the identified problematic Unicerts were issued by publicly trusted CAs. Overall, the Unicode- and IDN-specific issues remain outside the scope of existing certificate compliance checks.

For RQ2, we uncovered field decoding and character handling anomalies in all 9 tested TLS libraries (Section 5). Common decoding issues include incompatible (e.g., Forge decodes UTF8String with ISO-8859-1) or over-tolerant decoding methods (e.g., GnuTLS decodes PrintableString with UTF-8), which can cause inconsistent or incorrect extraction of entity information. Additionally, each TLS library exhibited at least one violation in handling special characters, such as improper escaping or accepting illegal characters beyond standard encoding ranges. These issues could enable subfield forgery, CRL spoofing, or validation bypass. However, the practical security risk is low, as the relevant threat models are uncommon and many can be mitigated by current PKI practices, such as discouraging CN-based hostname validation [9], deprecating BMPString [17], and replacing revocation checks with the promotion of short-lived certificates [8, 30, 57].

For RQ3, we showed that current noncompliance with Unicode can impact security and usability. Through empirical study, we revealed some threat surfaces: (i) misleading CT monitors to conceal specific forged certificates (Section 6.1); (ii) obfuscating certificate-based traffic analysis in TLS 1.2 and earlier (Section 6.2), and (iii) spoofing users into trusting unverified websites, though this requires stringent conditions and has limited practical impact (Appendix F.1).

For stronger mitigation, we responsibly disclosed issues to relevant entities and provided recommendations for the community to improve compliance when adopting multilingual text in PKI.

Contributions. Our main contributions include:

- We conducted the first systematic study on the Unicert issuance and parsing compliance.
- We identified three types of noncompliance with the Unicert issuance requirements and evaluated their prevalence via large-scale measurements.
- We revealed attribute decoding and Unicode character handling anomalies in 9 TLS libraries, demonstrating their potential to cause security and usability problems.
- We provided recommendations and tools to improve Unicert handling and help establish a global PKI system.

2 Background

2.1 Public Key Certificate Standards

In the PKI ecosystem, many standards govern the issuance of public key certificates to ensure security and interoperability. The X.509 standard [45] defines the structure and data types of these certificates. Building on X.509, the Public Key Infrastructure X.509 (PKIX) group stipulates a series of RFCs that specify Internet-relevant aspects of certificate management, such as formats, revocation, and path validation. At the core of PKIX is RFC 5280 [17], which defines the standard format for X.509 version 3 certificates used on the Internet. Subsequent updates (e.g., RFC 8399 [41], RFC 9549 [42], and RFC 9598 [63]) further refine how certificates handle Internationalized Domain Names (IDNs) and Internationalized Email Addresses (IEAs). Meanwhile, since domain names (DNSNames) often serve as primary identifiers in certificates, they must conform to syntax and formatting rules defined by DNS standards like RFC 1034 [65] and RFC 5890 [52]. In addition, the CA/Browser (CA/B) Forum, comprising major browser vendors and CAs, has developed Baseline

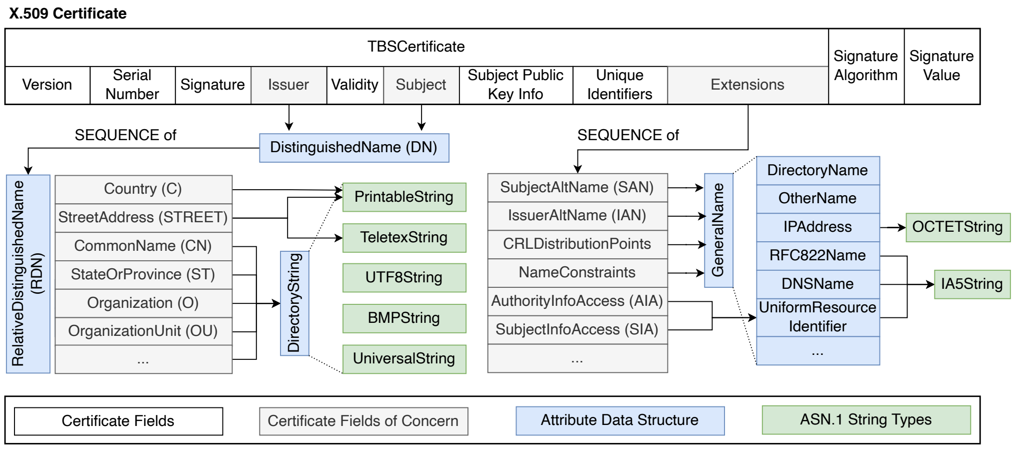


Figure 1: X.509 certificate fields, data structures, and encoding types. See detailed descriptions in Appendix B.

Requirements (BRs) [9] based on RFC 5280, ensuring that CA-issued certificates meet browser and client software requirements.

2.2 X.509 Certificates

Certificate encoding. An X.509 certificate is defined using ASN.1 (Abstract Syntax Notation One), a formal language for specifying data structures used in encoding and decoding [46, 47]. One of the standard encoding rules applied to certificates is DER (Distinguished Encoding Rules), a strict and deterministic binary format of ASN.1 that guarantees unambiguous data representation. In DER, each ASN.1 object is encoded as a *TLV* (*Tag*, *Length*, *Value*) triplet. Specifically, the *Tag* indicates the type of ASN.1 string used (e.g., PrintableString, IA5String, and UTF8String [76]); the *Length* defines the number of bytes in the content; and the *Value* carries the actual data, encoded according to the string type in *Tag*. Importantly, each ASN.1 string type has its own standard character sets and encoding rules. For clarity, we summarize the ASN.1 string types relevant to this paper in Table 8 (Appendix B).

Certificate fields. Each X.509 certificate includes three main components: TBSCertificate, Signature Algorithm, and Signature Value (Figure 1). The TBSCertificate (to-be-signed certificate) contains mandatory fields like Issuer, Subject, and extensions defining the certificate's intended usage and validation parameters. Among these fields, CommonName (CN) and SubjectAltName (SAN) are most commonly used to identify entities via domain names, IP addresses, email addresses, or uniform resource identifiers (URIs). In addition to identity, certain extensions play a critical role in certificate validation. For example, CRLDistributionPoints specifies where to retrieve certificate revocation lists, and AuthorityInfoAccess (AIA) provides information for accessing the issuing CA's certificate. These elements collectively support robust trust and validation mechanisms in public key infrastructures.

Data structures in certificates. A certificate includes various complex data structures. Key structures of certificate fields include GeneralName (GN), an ASN.1 type supporting identifiers like DNSName, RFC822Name, and IPAddress. Another core structure is DistinguishedName (DN), which identifies entities like issuers and subjects using a sequence of RelativeDistinguishedNames (RDNs). Each RDN holds one or more attribute type–value pairs, with some values encoded as DirectoryString types (e.g., UTF8String, PrintableString). The DirectoryName option in GeneralName embeds a full DN, while other forms like RFC822Name and DNSName encode email addresses and domain names using IA5String. These data structures collectively support rich identity representation and flexible naming in X.509 certificates.

2.3 Terminology

The standard terms related to PKI and X.509 certificates are provided in Appendix B. We define several key terms used throughout this work as follows:

- *Non-PrintableASCII characters*: All characters beyond the standard printable ASCII range of U+0020~U+007E, including control codes, multilingual scripts, and other Unicode blocks.
- *Unicert*: X.509 certificates that contain internationalized contents, such as IDNs, IEAs, and internationalized resource identifiers (IRIs), or multilingual text with characters beyond the standard printable ASCII set.
- *IDNCert*: The Unicerts containing IDNs in certificate fields.
- *Compliance / Noncompliance*: Cases that adhere to or violate specified standards or requirements (e.g., RFCs, CA/B BRs).
- *Misissuance*: The issuance of a certificate in violation of the rules set forth by the CA/B Forum, typically involving improper validation that results in certificates for unauthorized parties or with incorrect information.

2.4 Related Work

X.509 certificate measurement. The adoption of X.509 certificates has been widely studied. Prior studies have measured widespread misconfigurations of certificates, e.g., hostname mismatches and untrusted certificates, across websites, IoT, and email services [25, 26, 39]. These issues, along with shared certificate usage, can enable attacks such as hostname validation bypass and traffic hijacking [27, 37, 71]. However, these studies did not cover Unicert deployment in the shift toward internationalized PKI.

Certificate compliance. Early work by Delignat-Lavaud et al. emphasized the detection of CA/B compliance to improve certificate issuance [24]. Motivated by emerging reports of noncompliance or errors in CAs [81, 82], Kumar et al. developed Zlint to identify noncompliant certificates, discussing the historical misissuance landscape [53]. Over the years, the community has developed various certificate checking linters [6, 32, 80, 88, 96] for ensuring better issuance. However, these efforts aren't explicitly focused on Unicode handling and Unicert compliance.

Certificate parsing and validation. Much research has examined implementation flaws in certificate validation [7, 14, 22, 85], primarily focusing on hostname matching and chain-of-trust verification. These studies typically assume successful certificate parsing, overlooking the correctness of the parsing process itself. While some research has modeled ASN.1 parsing logic to develop accurate parsers [22, 23] and others have identified flaws in mainstream certificate parsers by mutating ASN.1 tree structures [13], they have not addressed the parsing of Unicerts with Unicode attributes and internationalized contents like IDNs.

3 Methodology

This study investigates whether the issuance and parsing of Unicert are compliant with current standards. This section first outlines our method for analyzing standard constraints and measuring Unicert issuance compliance, then details how we test parsing anomalies in general-purpose TLS libraries for assessing the potential implications of noncompliance.

3.1 Unicert Compliance Analysis

Assessing Unicert compliance requires understanding standard requirements across all certificate fields. Given the diverse field types and application scenarios, the requirements for Unicerts depend on prerequisites or references to external standards (e.g., RFC 5280 depends on DNS standards like RFC 1034), making comprehensive standard extraction necessary. This involves two key challenges:

Challenge 1: Extract normative constraints from multiple standards that are intricate and interdependent. First, the content specified in a standard would be continually updated and refined by new standards. For example, RFC 5280 defines rules for encoding core certificates, while updates like RFC 6818 specify details such as internationalized email addresses. Second, a standard may cross-reference other domain-specific standards. For instance, RFC 1034 defines the DNSName format, referenced in RFC 5280. The CA/B BRs further build on the structure established by RFC 5280. We must consider these iterative updates and references to extract normative constraints.

Challenge 2: Deal with complex formats, structures, and encoding rules of X.509 certificate fields expressed in various representations. Most requirements are written in natural language, while some are defined in ASN.1 using nested data structures and encoding types (e.g., UTF8String). These encoding types specify valid character ranges for fields, which can vary by context. For example, CAs typically encode DirectoryString with PrintableString or UTF8String, except when issuer fields use other encodings or when issuing certificates to existing subjects [17]. Although IA5String is equivalent to the ASCII range, it is restricted to include only [a-zA-Z0-9.-] in the context of DNSName. Understanding how these expressions correspond in context is essential for extracting accurate constraints.

3.1.1 *Analyzing specifications.*

Emerging pretrained Large Language Models (LLMs) have proven effective at analyzing natural language specifications [29, 48, 64, 99]. With sufficient background, LLMs can interpret context-dependent rules and nested structural constraints across various representations like natural languages, ASN.1 structures, and tables. To address the challenges of standard analysis, we employed RFCGPT [89, 90], a specialized GPT-4-based tool pretrained on ~2K RFCs and well-equipped to navigate complex requirements. We used it to analyze standards related to certificate profiles, including PKIX specifications (RFC 3280, 5280), their updates (RFC 9549, 9598, 6818), references (RFC 3490, 1034, 3454), dependent standards (RFC 6125, IDNA suites), and CA/B BRs. The analysis process involves:

Step I: Extracting relevant standards and incorporating references to reflect their evolution and interdependencies. We filtered field-related sections from the desired documents using a set of specified keywords[2]. We then refined the filtered sections by incorporating references from other specifications (e.g., domain name formats in RFC 1034) and replacing the outdated sections with updates from newer RFCs (e.g., replacing "Section 7.3" of RFC 5280 with "Update to RFC 5280, Section 7.3" in RFC 6818).

Step II: Improving LLM comprehension of background knowledge. Inspired by a prior *background-augmentation* method [61], we incorporated the refined sections as background knowledge to enhance RFCGPT's understanding of Unicert. Since CA/B BRs are not in RFCGPT's pretraining data, we added their certificate profile content as supplemental knowledge. All knowledge was extracted as line-based text and placed in the "background context" fields of the prompt templates (Figure 6, Appendix C). For example, rules listed in tables were reformatted into comma-separated lines.

Step III: Prompting the LLM to generate certificate field requirements. We used few-shot learning [64, 100] to prompt the LLM to extract two types of requirements for each certificate field: (1) valid encoding types and data structures, and (2) constraints on encoding and formatting. We excluded semantic constraints like field presence, absence, or criticality. Prompts included input–output examples to guide the LLM in generating requirements in a structured format (e.g., JSON), and directed it to reply "No" when standards lacked relevant details, avoiding fabrication. The prompt templates are detailed in Appendix C.

Our LLM-based approach generates 95 constraint rules for the data structures, encoding types, and valid character ranges for 36 Unicode-related fields. Through manual review, we confirmed the correctness of the identified constraints.

3.1.2 *Checking issuance compliance.*

To assess Unicert issuance compliance, we built a certificate linter to verify its contents and format against standard requirements. Among established linters [6, 32, 88, 96], which vary in standards coverage and integration options (e.g., CLI, APIs), we selected Zlint [96] for its: (1) broad rule

[2]The keywords we used include ASN.1 encoding types, "encode/decode", "character", "string", "internationalized", "Unicode/ASCII/UTF8", "NFC", "IDN", and "IRI".

set (594 lints) covering RFC 5280 and CA/B BRs, (2) open-source and easily extensible Go codebase, and (3) active maintenance.

Since Zlint is not specifically tailored for Unicert compliance, it includes 45 lints related to character ranges and field structures, which are limited to direct constraints explicitly defined in standard documents (e.g., maximum field length, empty values). For the remaining 50 constraints identified in Section 3.1.1, we implemented custom lints within the Zlint framework and added them to the lint registry accordingly. These new lints handle more nuanced cases. For example, in internationalized email support, RFC 5280 allows RFC822Name fields (email addresses) to contain IDNs, but updates like RFC 9598 [63] restrict RFC822Name to US-ASCII, requiring SmtpUTF8Mailbox for non-ASCII local parts and IDNA2008-compliant LDH labels for domain parts [52].

Notably, Zlint allows customizing issue severity (e.g., warning, error) per requirement levels defined in standards (e.g., SHOULD, MUST), helping us effectively assess the impact of noncompliance. We also assigned effective dates to each lint, specifying the period during which a rule applies to newly issued certificates, to avoid retroactively applying new rules to older Unicerts. This approach makes the detected noncompliant cases a lower bound, since certificates issued before the effective date may still pose risks if they remain valid and are not revoked in time.

After implementing the lints, we rebuilt the tool and applied it to the Unicert dataset we collected (see details in Section 4.1) to identify noncompliant certificates. The names, sources, and severity of the employed lints are listed in Appendix D, and the noncompliance types covered by these lints are analyzed in Section 4.3.1.

3.2 TLS Library Analysis

To assess the potential impact of noncompliant issuance, we tested parsing issues in 9 general-purpose, open-source TLS libraries that are widely studied in prior work [7, 22, 23, 85] and offer developer-friendly certificate parsing APIs. The full lists of the tested libraries and APIs are detailed in Appendix E. The testing involved (i) generating test Unicerts and (ii) analyzing parsing compliance for key fields (i.e., Subject, Issuer, SAN, IAN, AIA, SIA, and CRLDistributionPoints). During the parsing analysis, we inferred the decoding methods and character checks based on the parsing results.

Selecting target parsing APIs. Given the complexity of parsing X.509 certificates, TLS implementations often rely on lower-level cryptographic libraries (e.g., ASN.1 libraries). We focused on APIs capable of directly parsing built-in string types, excluding functions that require custom use of ASN.1 libraries. Hence, we reviewed relevant source code and documentation of the libraries, selecting developer-accessible APIs as test targets.

Generating test Unicerts. To uncover parsing anomalies across APIs, we built a generator to craft diverse test Unicerts, as existing CT logs lack sufficient variation for corner cases. The generator creates Unicerts following the three rules: (i) simplify ASN.1 structures by using one RDN per DN and one attribute per RDN, (ii) generate random attribute values by inserting special Unicode characters (e.g., those outside standard ranges), and (iii) mutate only one field per Unicert while keeping other required fields at normal default, standard-compliant values (e.g., "test.com" for DNSName).

To test how parsing APIs handle special Unicode, we generated random attribute values by embedding Unicode into preset normal defaults. Given the large Unicode space, we sampled all characters from U+0000~U+00FF (due to RFC constraints [51, 98, 102] and known vulnerabilities [59, 70]), and one character from each of 323 standard Unicode blocks (excluding surrogates) [21]. In addition, to test decoding behavior during parsing, we also varied attribute types across all ASN.1 string types permitted in X.509 certificates. Appendix E outlines the specific ASN.1 string and field types used.

Analyzing parsing compliance. Assessing whether a TLS library parses Unicerts compliantly involves two questions: (1) Does it decode Unicode fields using standard methods? (2) Does it handle special characters according to RFC recommendations [51, 98, 102]? Due to the complexity of the process, we analyzed how libraries parse various Unicert fields to infer their decoding methods and character checks. For each test field, we generated multiple Unicerts and processed them using the target APIs to obtain parsing results. Through manual analysis of sampled results, guided by relevant RFCs [51, 98, 102] and historical CVEs [59, 70], we summarized five common decoding methods: ASCII, ISO-8859-1, UTF-8, UCS-2, and UTF-16, and three special character handling modes: character truncation, character replacement, and character escaping. We then inferred the decoding method and whether character checks were applied based on the parsing results. Cases with complete parsing failures (i.e., no output) were excluded from this inference and analyzed separately via manual inspection.

Inferring decoding methods. We decoded each Unicert field using the five common decoding methods. If an API's parsing results consistently match one method, we infer that the API uses that method. However, APIs may additionally process special characters. For example, the final parsed value would be "test\\x01\\xFF.com" after character escaping of the decoded value "test\x01\xFF.com". To cover such discrepancies, we further applied the three special character handling modes to the tested Unicerts after the five decoding methods, and rechecked whether the API could match them. If matched, we used that decoding method as the inferred result and recorded the character checking method for further analysis.

Inferring character checking methods. We then systematically analyzed TLS libraries' character-checking methods and parsing errors, identifying three types of anomalies: (1) Byte sequences that should have triggered decoding errors were not properly handled. (2) Parsed characters violated the ASN.1 standard specifications. (3) Character escaping did not comply with the relevant standard requirements. We provided an in-depth analysis in Section 5.2.

3.3 Limitations

Our methods have several limitations: (1) The study aims to evaluate the Unicert ecosystem rather than deliver a fully automated tool. Thus, some rules still require manual validation and adjustment, limiting scalability and reproducibility. While LLM-based and library examination methods help uncover broader compliance issues, they serve as complementary tools rather than standalone solutions and still depend on expert oversight. (2) Keyword filtering for LLM inputs may miss descriptive information not explicitly tied to those terms, like DN comparison steps in RFC 5280 or table footnotes in the BRs. We tried to review the documents to ensure that no information on common fields was omitted. (3) Certain compliance checking rules depend on certificate issuing history, such as the use of TeletexString being allowed only for previously established Subjects. However, the current linter setup operates on a single certificate at a time and does not support cross-certificate validation, making it inadequate for such context-sensitive checks. (4) Our Unicert parsing analysis focuses on general-purpose open-source libraries, as their accessible code enables precise instrumentation and in-depth analysis of their handling of certificate fields. Tools like browsers, IDSes, or email clients often use proprietary parsers, making them opaque for scalable testing.

4 Compliance of Unicert Issuance

This section investigates Unicert issuance compliance to infer CA support for internationalized contents (e.g., IDNs, multilingual text) and reveal the noncompliance landscape and problematic practices.

4.1 Certificate Dataset

Dataset collenction. Our analysis relies on Certificate Transparency (CT) logs, which provide transparency and verifiability for certificate issuance from the most prominent CAs. We collaborated with a security company, QiAnXin, which has collected over 70 billion certificates from 16 CT logs [33]. The selected log list is mainly maintained by Google, which claims to limit the accepted root certificates according to the trust store [15]. In all CT entries, about 54.7% are precertificates, which are only used to verify certificate validity before issuance and should not be deployed [55]. So we filtered out the precertificates by checking their unique CT Poison extensions, leaving 32 billion regular certificates.

From the regular set, we further extracted 34.8 million leaf certificates containing characters beyond Printable ASCII (0x20~0x7e) in any fields or containing IDNs in the DNSName-related fields (e.g., CommonName and the extensions shown in Figure 1). The CT dataset includes a substantial number of historical certificates, with 7,621,405 Unicerts remaining valid as of the final analysis month, April 2025. To assess Unicert compliance over time, we retained all historical data and assigned effective dates to each checking lint, enabling evaluation of CA behavior across different periods.

Limitations. The dataset may be biased due to incomplete CT log coverage and potential loss during data crawling. Additionally, since the CT program only began in 2015, Unicerts issued before then may be underrepresented, though they still offer insight into historical practices. Despite these limitations, the dataset is large enough to reveal common issues across the ecosystem.

4.2 Unicert Issuance Overview

Unicert support across issuers. The Unicerts in our dataset were issued by 4,528 CA certificates across 698 issuer organizations (the IssuerOrganization field), covering 87 trusted CA owners listed in CCADB [66]. A few major issuers with organization names of "Let's Encrypt" (25.1M), "COMODO CA Limited" (4.8M), and "cPanel, Inc." (1.3M) account for 89.4% of Unicert issuance, exhibiting an oligopoly pattern. However, compared to common website certificates [53], Unicert issuers show greater diversity, with many regional and national organizations addressing localized needs and supporting region-specific scripts, such as "GEANT Vereniging" (215K) and "DOMENY.PL sp. z o.o." (49K).

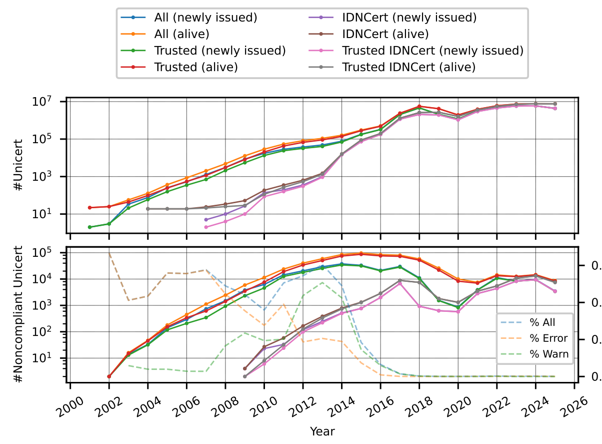


Figure 2: Issuance trend of Unicerts and noncompliant Unicerts (as of April 2025). The *alive* lines indicate Unicerts that remain valid in each year. Though a log scale may obscure value differences, it is essential for showing trends across a wide range.

Historical Unicert issuance. Figure 2 shows a clear upward trend in Unicert issuance. The patterns for all and trusted Unicerts closely align, as over 97.2% of newly issued Unicerts each year since 2015 have come from trusted CAs[3]. This reflects growing alignment with trusted root programs and broader CA adoption. In total, 31,339,050 (90.1%) were historically issued by trusted CA owners. This high trust rate likely stems from strict CA admission policies enforced by the CT providers [11, 15, 55]. Moreover, distrusted CAs may either refrain from CT submission or cease certificate issuance after losing trust. Accordingly, the following analysis focuses on trusted Unicerts unless noted otherwise.

4.3 Noncompliant Unicert Landscape

Using our certificate linter on CT-logged Unicerts, we identified 249,281 Unicerts (0.7%) as noncompliant[4]. Of these, 162,789 (65.3%) were issued by publicly trusted CAs, while 52,544 (21.1%) were by providers with limited trust, recognized only in certain regions (e.g., governmental CAs), scenarios (e.g., cloud services), or user agents (e.g., region-specific browsers). To highlight key issues, we first summarized common issue types, then examined noncompliant issuance over time.

4.3.1 *Noncompliance taxonomy.*

To better understand the nature and causes of noncompliant issuance, we classified the noncompliant Unicerts into three types. Table 1 summarizes all types with

[3]For longitudinal analysis, we treat certificates as trusted if their issuers were trusted at the time of issuance, ignoring later CA deprecation or acquisition.

[4]Note that if the effective dates of the certificate lints are ignored, the result increases to 1.8 million, indicating many certificates issued before the effective dates still have issues. However, we do not consider them noncompliant in this study.

Table 1: Overview of noncompliance types.

Noncompliance Types		#Lints		#NC Unicerts						
		All (New)	NC (New)	All lints	New lints	Error	Warning	Trusted	Recent	Alive
T1	Invalid Character	22 (10)	15 (6)	43.2K (17.3%)	26.9K (62.1%)	42.2K (97.6%)	1.7K (4.0%)	94.7%	11.6K (26.9%)	14.6K (33.8%)
T2	Bad Normalization	4 (3)	1 (1)	3 (0.0%)	3 (100%)	3 (100%)	0 (0.0%)	100.0%	0 (0.0%)	0 (0.0%)
T3	Illegal Format	17 (0)	8 (0)	3.2K (1.3%)	0 (0.0%)	3.2K (100%)	0 (0.0%)	47.0%	13 (0.4%)	36 (1.1%)
	Invalid Encoding	48 (37)	23 (18)	150.9K (60.5%)	56.3K (37.3%)	58.7K (38.9%)	117.5K (77.9%)	69.7%	12 (0.0%)	114 (0.1%)
	Invalid Structure	2 (0)	2 (0)	93.7K (37.6%)	0 (0.0%)	93.7K (100%)	0 (0.0%)	49.0%	1.4K (1.5%)	3.5K (3.8%)
	Discouraged Field	2 (0)	1 (0)	589 (0.2%)	0 (0.0%)	0 (0.0%)	589 (100%)	18.8%	1 (0.2%)	4 (0.7%)
All		95 (50)	50 (25)	249.3K (100%)	83.1K (33.3%)	183.9K (73.8%)	118,958 (47.7%)	65.3%	13.9K (5.2%)	18.1K (7.3%)

[1] NC indidate noncompliance-related.
[2] "Trusted" indicates globally trusted NC Unicerts; "Recent" refers to those issued in 2024~2025; "Alive" shows those still valid in 2024~2025.
[3] Since a Unicert can fail multiple lints, the sum of #Error and #Warning exceeds the number of unique noncompliant Unicerts.
[4] Percentages in the "All lints" column are based on all NC Unicerts (249.3K), while other percentages refer to the NC Unicerts within each subtype.

covered lints, severities, affected Unicerts, and trusted rates to highlight their potential impact. We discuss each type in detail below.

T1. Character range inspection. Some CAs perform inadequate checks on character ranges for certificate field value, resulting in (i) malformed strings (e.g., non-printable characters in PrintableString), or (ii) disallowed characters (e.g., control characters) in UTF8String. We categorized these issues as *Invalid Character*.

Table 1 shows 22 lints used for this type, identifying 43,240 noncompliant Unicerts, mostly with error-level violations, with 26.9% newly issued and 33.8% valid in 2024~2025. Some (13.9K) include NUL, ESC, or DEL in Subject attributes, triggering the lint of `subject_dn_not_printable_characters`. This type of issue can hinder the parsing, validation, and display of entity information for Unicerts. For example, studies showed that NUL byte terminations can cause misinterpretation of CNs during hostname verification [49, 85]. Additionally, visual rendering and traffic inspection based on these fields can be compromised, referring to Section 6.

T2. Field value normalization. Value normalization is essential for matching distinguished names, particularly during name chaining. It requires UTF-8 strings to use canonical composition form (NFC) [35, 78], and IDN U-labels (Unicode) to be converted to A-labels (Punycode) for comparison and storage [17, 42], then back to Unicode for display. However, some IDN A-labels are malformed, either unconvertible to U-labels or containing characters beyond the domain standard after conversion. This complicates matters since implementations must convert IDNs to Unicode before displaying and comparing certificates. Meanwhile, non-conforming IDNs introduce the possibility of conversion errors between alternate forms [63]. We categorized these issues as *Bad Normalization*.

This type is the least common in our dataset, mainly involving IDNs not normalized to NFC after converting Punycode to Unicode. Despite its low frequency, this issue is important, as past vulnerabilities showed malformed Punycode email addresses could trigger buffer overflows and denial-of-service attacks [10, 73].

T3. Field format and structure checks. We classify violations related to the format or structure of certificate fields into four subtypes: (1) *Illegal Format*: basic formatting errors, such as length overflows or incorrect character cases, which can hinder parsing. (2) *Invalid Encoding*: use of unsupported encoding types, e.g., encoding CN with TeletexString or BMPString instead of IA5String. (3) *Invalid Structure*: violations of structural rules, such as required inclusion (e.g., CN must appear in SAN) or duplicate attributes (e.g., multiple CNs in Subject). (4) *Discouraged Field*: use of non-recommended fields (e.g., CN in Subject or URI in SAN).

Cases involving invalid encoding represent the largest share (60.5%) of all noncompliant issuances. Notably, 22.6% of noncompliant Unicerts were detected by our new lints, suggesting that encoding issues of Unicerts have been under-addressed by the community. Violations of format and structure requirements can disrupt certificate usability (e.g., causing connection failures) or prevent accurate identity extraction. For instance, PyOpenSSL (v.24.2.1) selects the first CN, while Go Crypto (v.1.23.0) uses the last when handling duplicated Subject attributes. In addition, current standards do not strictly prohibit discouraged attribute types (e.g., CN), but continued issuance of such certificates by CAs can complicate entity identification from certificates.

4.3.2 Longitudinal view of noncompliance.

Although many Unicerts are noncompliant, the overall noncompliance rate remains low (0.7%). Figure 2 shows great improvement in Unicert compliance since 2015. This trend reflects growing attention to issuance compliance, driven by extensive studies on noncompliance [24, 53, 81, 82], the enforcement of CT, and the release of certificate linters [6, 32, 80, 88, 96]. Despite this progress, many CAs still engage in noncompliant practices, and the issued Unicerts can be valid until 2050.

Issuers associated with noncompliant Unicerts. All of the identified noncompliant Unicerts were issued by 2,922 certificates across 505 issuer organizations, covering 78 CA owners in CCADB and 295 untrusted or unknown issuers. We found issuance flaws are widespread, involving both globally prominent and regional CAs. Unlike normal Unicerts, noncompliant Unicerts are more evenly distributed across organizations, showing no clear oligopoly.

Table 2 shows organization names with the most historical noncompliance. Some publicly trusted issuers also show high rates, suggesting public trust does not ensure strong Unicert compliance. This may stem from CA lint tools not yet fully covering the internationalization requirements. Meanwhile, issuers with over 80% noncompliance suggest possible systemic issues.

In contrast, the top 10 issuers with the highest Unicert volumes show low noncompliance, each below 6% and about 2% overall. Some issuers, such as Let's Encrypt, Cloudflare, and Amazon, issue only IDNCerts, likely due to automated domain validation workflows that limit field customization (e.g., Let's Encrypt permits only DNSNames and filters out non-domain characters). Such constraints may help reduce noncompliance.

Table 2: Top 10 issuer organization names by noncompliant Unicerts.

Issuer OrganizationName	Trust Status[2]	Region	Non-compliant[3]	Recent[4]
Česká pošta, s.p.⋆[1]	◐	CZ	22,939 (96.39%)	0
Symantec Corporation	○	US	18,092 (51.47%)	0
Dreamcommerce S.A.	●	PL	17,291 (44.83%)	0
DigiCert Inc	●	US	17,276 (3.40%)	40
Let's Encrypt	●	US	15,484 (0.06%)	7,091
StartCom Ltd.	○	IL	14,168 (72.97%)	0
COMODO CA Limited	◐	GB	11,870 (0.25%)	0
ZeroSSL	●	AT	11,224 (2.53%)	4,094
Government of Korea	◐	KR	10,416 (87.33%)	0
VeriSign, Inc.	●	US	7,513 (59.12%)	0
Other	-	-	103,008 (0.29%)	1,802
Total	-	-	249,281 (0.72%)	13,027

[1] Long names are abbreviated with ⋆ for brevity.
[2] ●: publicly trusted; ◐: trusted in specific regions or scenarios; ○: not trusted.
[3] Percentages based on each organization's total Unicerts.
[4] Noncompliant Unicerts signed in 2024 and 2025.

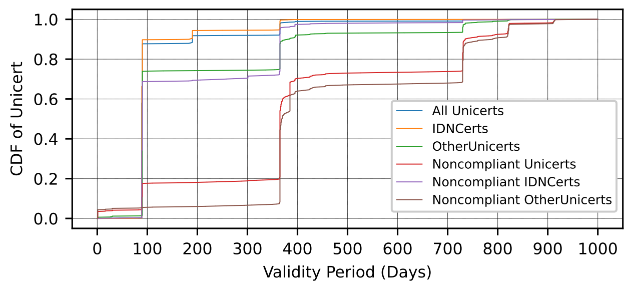


Figure 3: CDF of Unicert validity period. The CDF's long tail is truncated beyond 1,000 days of validity.

Validity of noncompliant Unicerts. Figure 3 shows validity period differences across Unicert types. Among them, IDNCerts showed better compliance with modern standards, with 89.6% conforming to the 90-day validity trend [93]. In stark contrast, other Unicerts often exceed the recommended 398-day validity [9], with over 10.7% lasting longer. Compared to normal Unicerts, the noncompliant ones tend to have much longer lifespans, with about 50% lasting a year and over 20% exceeding 700 days. These extended lifespans may reflect a lag in Unicert's compliance with modern standards and increase security risks by prolonging the window for potential compromise or exploitation.

4.4 Troublesome Certificate Fields

To evaluate the most troublesome certificate fields, we analyzed 21 fields that permit non-ASCII input and found wide variation among issuers. Most allow non-ASCII Unicode or IDNs in Subject/Issuer, and some extend this to extensions like SubjectAltNames and CertificatePolicies, which are typically related to human interaction and user identity information. However, some issuers also show noncompliance in handling certain Unicode fields (Figure 4). Common issues include discouraged CN use, repeated Subject attributes, and incorrect encoding types. We categorize the most troublesome fields into three main groups, summarized below.

Weak validation of entity name fields. Public key certificates bind entity identities to keys, making accurate entity name parsing critical. Errors can create ambiguity in identifying and verifying peer entities, especially in CN and SAN fields (DNSNames, EmailAddresses, URIs) [84]. However, many issuers still inadequately validate these fields. The flaws include:

[F1] *Poor validation of DNSNames.* We found 51 issuers that have issued Unicerts with noncompliant DNSNames. Specifically, we identified 27,102 cases under the invalid character type in Table 1, containing malformed IDNs that either (i) cannot convert to Unicode or (ii) include illegal characters (e.g., bidirectional controls) after Punycode decoding, violating the IDNA standard [28]. These IDNs are only syntactically valid (e.g., starting with "xn--") and can resolve via wildcard DNS, though the specific subdomain labels may not exist. Results showed that this issue persists, affecting prominent CAs such as Let's Encrypt and Sectigo. Our discussions with Let's Encrypt and Mozilla Bugzilla suggest the root cause lies in complex standard interdependencies and a narrow CA focus on domain control validation per CA/B BRs (see Section 7).

[F2] *Invalid values in CNs and SANs.* Though deprecated, CNs remain widely used (e.g., Snort [87], cURL [19], Postfix [77]) due to fallback needs when SANs are missing. We found 27,478 Unicerts containing invalid characters in the CN or SAN fields (see the invalid character lints in Appendix D). For example, we found CN fields containing U+0000, U+202E, U+000A, and a SAN field containing an entire CSR PEM string. This issue is less common in pure IDNCerts since their Unicode IDNs have been converted to Punycode.

Error handling in distinguished name (DN) fields. Each DN uniquely identifies an entity in a certificate, supporting (i) certificate chaining, (ii) CA decisions on issuance, and (iii) entity information display in applications like browsers and email clients. Poor DN attribute validation can confuse both programs and users when inspecting entities. However, we found 14,722 Unicerts with DNs containing invalid characters, potentially causing rendering or usability issues. The common flaws include:

[F3] *Certificate fields shown in browser or software UIs are especially error-prone.* The CN, O, and OU fields are commonly visualized in user agent software. Our analysis reveals major issues in these Subject attributes, as shown in Figure 4. Their encoding and rendering directly impact user trust or aid attackers in spoofing attempts (see Appendix F.1 for examples).

[F4] *Potential software defects or bugs.* We found 117 cases with DN fields containing `DEL` (U+007F). In some cases, deleting characters based on `DEL` count reveals meaningful text. For exmaple, the Subject field value "Prepard`[DEL][DEL]`id Serc`[DEL]`vices" could represent "Prepaid Services". Some reports suggest locale issues may cause the delete key to input a `DEL` character instead of removing a character [3]. However, we cannot directly attribute this issue to a CA software bug, as it is not limited to a few CAs - these Unicerts were signed by 20 organizations across 8 regions. Additionally, some CAs added special characters at "regular intervals" due to implementation bugs. We found 400 Unicerts with evenly inserted `NUL`s (e.g., "`[NUL]`C`[NUL]`&`[NUL]`I`[NUL]`S" rendered as "C&IS"). Such Unicerts were mainly issued by IPS CA and Thawte Consulting, likely due to their issuing implementations.

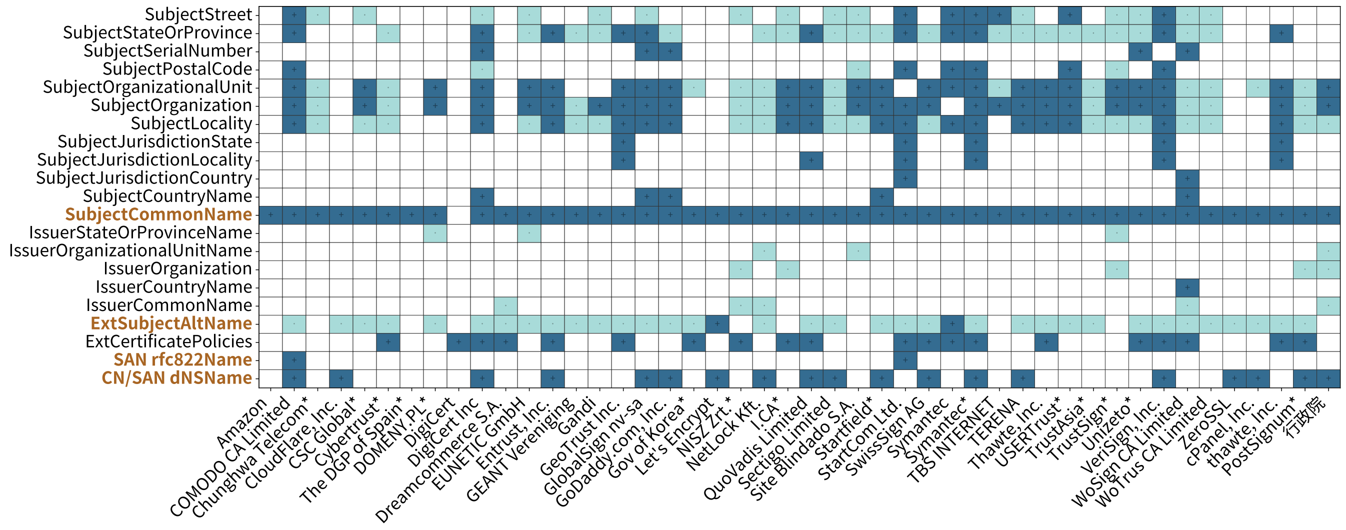


Figure 4: Fields containing internationalized contents and characters beyond U+0020~U+002E. We included CAs (issuer organization names) signing over 5K Unicerts. Long organization names are abbreviated with an asterisk (*) for brevity. Colored (·/+) fields show Unicode, with the darkest (+) indicating deviation from standards.

Table 3: Value variant strategies in Subject fields.

Variant Strategy	Variant Examples
Character case conversion	Samco Autotechnik GmbH SAMCO Autotechnik GmbH
	NOWOCZESNASTODOŁA.PL SP. Z O.O. nowoczesnaSTODOŁA.pl sp. z o.o.
Abbreviation variations	SKAT ELEKTRONIKS, OOO SKAT Elektroniks Ltd.
	RWE Energie, s.r.o. RWE Energie, a.s.
Addition of non-printable characters	PEDDY[U+00A0]SHIELD[U+00A0]... Peddy Shield ...
Use of different whitespace characters	株式会社 [U+0020] 中国銀行 株式会社 [U+3000] 中国銀行
Substitution of resembling	EDP -[U+002D] Energias de Portugal, S.A EDP –[U+2013] Energias de Portugal, SA
	Vegas.XXX®[U+2122] (Vegas...LLC) Vegas.XXX™[U+00AE] (Vegas...LLC)
	crossmedia:team GmbH crossmedia Team GmbH
Replacement of illegal characters	St[U+FFFD]ri AG (TeletexString) Störi AG (UTF8String)

Obfuscation in Subject fields. The Subject field is vital in several scenarios. CAs use it to check applicant information against blacklists before issuance and to revoke certificates linked malicious subjects. Similarly, security tools rely on it to match malicious entities against preset rules and threat intelligence. However, UTF8String's broad character support allows identity-equivalent certificates with mismatched Subject DNs, introducing potential detection evasion:

[F5] *CAs allow Subject variants without strict validation.* We found that Subject variants are common in CT logs. We identified six variation strategies, such as case changes and character tweaks, and summarized them in Table 3. For instance, the CountryName field of Germany, which should be a 2-letter PrintableString, appears as "Germany", "DE,de", "DE,DE", "GERMANY", and others. We also found multiple encodings of a French region name presented in the StateOrProvinceName field, such as "\u00c3\u0008le-de-France", "\u200e\u00cele-de-france", "\u00eele-de-France", all of which should be "Île-de-France". These variants can complicate Subject-based identity matching and enable malicious actors to evade CA scrutiny and hinder threat detection.

5 Analysis of TLS Implementations

Our measurement shows CAs often permit broader Unicode than standards allow, raising concerns about how TLS implementations handle invalid characters or encodings. This section examines whether popular libraries respect declared encodings and enforce strict character checks.

5.1 Attribute Decoding Issues

RFC 5280 requires TLS implementations to decode certificate attributes according to ASN.1-specified encoding types and corresponding character ranges [46, 47]. For instance, PrintableString and IA5String should only be decoded with ASCII. However, we found TLS libraries often apply default decoding rules, ignoring the types declared in the ASN.1 certificates. To verify their decoding compliance, we crafted Unicerts with mixed character ranges and ASN.1 string types (e.g., inserting non-ASCII characters into PrintableString) and applied the methods in Section 3.2, complemented by manual inspection, to infer decoding behavior.

Attribute decoding issues. Table 4 highlights three noncompliant decoding practices we uncovered across the libraries:

(1) *Incompatible decoding*: Occurs when a method mismatches the standard encoding, e.g., decoding BMPString with ASCII or decoding UTF8String with ISO-8859-1. We observed this practice in OpenSSL, Forge, and Java.security.cert.

Table 4: Decoding methods for DN and GN.

Encoding Scenarios	Decoding Methods	OpenSSL	GnuTLS	PyOpen-SSL	Crypto-graphy	Golang Crypto	Java.security.cert	Bouncy-Castle	Node.js Crypto	Forge
PrintableString in Name	ISO-8859-1	○	○	⊘	○	○	○	⊘	⊘	⊘
	UTF-8	○	⊘	○	⊘	○	⊘	○	○	○
	Modified ASCII	⊙	○	○	○	○	⊙	○	○	○
IA5String in Name	ISO-8859-1	○	-	⊘	○	○	○	⊘	⊘	⊘
	UTF-8	○	-	○	⊘	○	⊘	○	○	○
	Modified ASCII	⊙	-	○	○	○	⊙	○	○	○
BMPString in Name	ASCII	⊗	○	○	○	○	⊗	○	○	○
	UTF-16	○	⊘	○	⊘	⊘	○	⊘	○	⊘
	Modified ASCII	⊙	-	○	○	○	⊙	○	○	○
UTF8String in Name	ISO-8859-1	○	○	○	○	○	○	○	○	⊗
	Modified ASCII	⊙	○	○	○	○	○	○	○	○
IA5String in GN	UTF-8	-	⊘	⊘	⊘	⊘	○	-	○	○
	ISO-8859-1	-	○	○	○	○	○	-	○	⊘
	Modified ASCII	-	○	⊙	○	○	⊙	-	⊙	○

Each encoding scenario, involving multiple certificate fields and APIs, may correspond to different decoding methods.
Java.security.cert's BMPString parsing is ASCII-compatible, though its decoding behavior is unclear.
- The implementation does not support parsing relevant fields, so we ignore its decoding checks.
○ No decoding errors; ⊘ Over-tolerant decoding exist; ⊗ Incompatible decoding exist; ⊙ Modified decoding exist.

(2) *Over-tolerant decoding*: Expands character ranges beyond the standard, e.g., decoding PrintableString with ISO-8859-1 or BMPString with UTF-16, often seen when parsing DN and GN. For instance, GnuTLS uses UTF-8 to decode all ASN.1 string types (except BMPString) in DN and GN.

(3) *Modified decoding*: Handles undecodable bytes by replacing them with substitute characters. The substitution behaviors vary across libraries. For example, Java.security.cert replaces non-ASCII bytes with U+FFFD in DN and GN; PyOpenSSL converts them to U+002E (".") in GN within CRLDistributionPoints; and OpenSSL uses hexadecimal escape sequences (e.g., "\x2e\x4d").

Impact of attribute decoding issues. To assess real-world exploitability, we analyzed CT logs for encoding practices and identified 7,415 Unicerts with ASN.1 encoding errors. After reconstructing certificate chains via AIA extensions and verifying signatures, we found 5,772 were issued by trusted CAs. Among these, 150 had encoding errors in Subjects, 110 had errors in SANs, and 5,575 had errors in CertificatePolicies. These results indicate that ASN.1 string encoding is not strictly enforced from the CAs' perspective, leaving potential for exploiting decoding flaws.

As both encoding and decoding may deviate from standards, mismatches between encoded and decoded values are inevitable. They can arise in two ways: correctly encoded Unicerts may produce mismatches if implementations perform incompatible decoding, and non-conforming Unicerts can cause mismatches through over-tolerant decoding. Below, we discuss the impact of the mismatches from both security and usability perspectives:

(1) *Hostname validation bypass.* A potential security threat from encoding-decoding mismatches is hostname validation bypass. In this threat model, the attacker is a compromised or malicious CA aiming to perform a Man-in-the-Middle (MITM) attack by issuing a forged certificate that is difficult for domain owners or auditors to detect. The target is a client with flawed decoding logic (e.g., incompatible decoding) during certificate parsing. The attack succeeds when the client misinterprets a carefully encoded field with a hostname. For example, a Subject CN[5] encoded as a BMPString (Unicode) may be incorrectly decoded by a client expecting ASCII, transforming "\u6769\u7468\u7562\u792e\u636e" into "githube.cn". The likelihood of this threat scenario is low, as it requires both a flawed issuer and a flawed parser. While we identified certificates in CT logs that could cause such misinterpretation, we found no evidence that they have been exploited in attacks.

(2) *Field information misrecognition.* Beyond hostname validation, certificate fields also support tasks like service log auditing and threat hunting. Incompatible or over-tolerant decoding can cause encoding-decoding mismatches, leading to misinterpretation of field information. The possible threat model and impact depend on the certificate usage context. In log auditing, network adversaries could initiate TLS connections with malformed certificates, making the network logs hard to analyze [69]. In threat hunting, in-path network attackers could exploit mismatches to bypass field comparisons, similar to the threat model in Section 6.2.

(3) *Parsing failures.* Invalid bytes in certificate fields can trigger parsing failures (e.g., "*asn1: syntax error: PrintableString contains invalid character*") when decoded with incompatible methods, potentially occurring before other validation errors such as hostname mismatches or untrusted CAs. This scenario primarily leads to usability (not security) issues, specifically TLS connection termination, which renders the website temporarily inaccessible. The direct impact is a disruption of service availability, rather than a compromise of data integrity or confidentiality. In this sense, the outcome is similar to an in-path attacker disrupting a TLS connection. Our experiments show that OpenSSL, Java.security.cert, and Forge exhibit incompatible decoding errors, but their modified decoding methods prevent such failures.

5.2 Character Checking Errors

TLS implementations should also enforce character checks after decoding. First, special characters within valid ranges should be escaped according to standards [51, 98, 102]. For instance, commas (,) and plus signs (+) in CNs should be escaped. Second, each ASN.1 string type has a defined character set, as listed in Table 8 (Appendix B). Characters outside this set are considered noncompliant and should either be escaped or trigger appropriate errors.

[5] Although using the Subject CN field for hostname validation is not recommended, some software still relies on it.

Table 5: Standard violations in parsing DN and GN.

Standard Violations		Crypto-graphy	PyOpen-SSL	Golang Crypto	Bouncy-Castle	Java.secu-rity.cert	Node.js Crypto	Forge	OpenSSL	GnuTLS
Illegal chars in DN	PrintableString Violations	⊙	⊙	○	⊙	⊙	⊙	⊙	⊙	⊙
	IA5String Violations	⊙	⊙	○	⊙	⊙	⊙	⊙	○	-
	BMPString Violations	⊙	○	⊙	⊙	-	○	-	-	⊙
Illegal chars in GN	IA5String Violations	⊙	⊙	⊙	-	⊙	○	⊙	-	⊙
Non-standard escaping in DN	RFC2253 Violations	-	-	-	○	○	○	-	⊗	-
	RFC4514 Violations	○	-	-	⊙	⊙	○	-	⊗	○
	RFC1779 Violations	-	-	-	⊙	⊙	⊙	-	⊗	-
Non-standard escaping in GN	RFC2253 Violations	-	⊗	-	-	-	⊙	-	-	-
	RFC4514 Violations	-	⊗	-	-	-	⊙	-	-	-
	RFC1779 Violations	-	⊗	-	-	-	⊙	-	-	-

○ No standard violation; ⊙ Unexploited violations; ⊗ Exploited violations. "-" indicates not considered in this test and the reason is listed in Appendix E.

Character checking issues. During character checks, we observed standard violations across the tested TLS libraries. As shown in Table 5, none of the libraries enforced checks for illegal characters among all ASN.1 string types, and 5 exhibited character-escaping violations when converting certificate fields into X.509-text representation (e.g., converting a SAN extension with two subfields, DNSName="a.com" and DNSName="b.com", to "DNS:a.com DNS:b.com").

Impact of character checking issues. In our CT dataset, we identified 43,240 Unicerts containing invalid characters. Improper handling of the special characters in such Unicerts could have the following two impacts:

(1) *Certificate attribute forgery.* Allowing non-DNS characters in a DNSName within the CN and SAN fields can enable attribute embedding, leading to attribute forgery issues. For example, a Unicert with DNSName="a.com DNS:b.com" may be converted as "DNS:a.com DNS:b.com" in the X.509 text presentation, causing string-based analyzers to misinterpret it as valid for two domain names. This scenario enables a threat model in which a compromised or malicious CA crafts a Unicert with malformed attributes. Attackers can exploit inconsistencies in attribute handling across different components (e.g., clients and middleboxes) to bypass detection. We observed the issues in PyOpenSSL's GN parsing and OpenSSL's DN parsing, which allow injecting DNSNames into SANs or CNs into DNs using strings that embed additional attributes. This model resembles an existing vulnerability [70], which showed that ambiguous field transformations can be exploited to bypass certificate verification or name constraint checks.

(2) *Special character replacement.* When decoding GeneralNames in CRLDistributionPoints, PyOpenSSL replaces the control characters in ranges U+0000~U+0009, U+000B, U+000C, U+000E~U+001F, and U+007F with U+002E. This behavior enables a malicious entity that has compromised a CA's issuing infrastructure to effectively disable revocation by embedding control characters in a CRL location (e.g., "http://ssl\u0001test.com"), which a vulnerable client parser transforms into a different address ("http://ssl.test.com"). This threat model assumes the attacker has no control over the CA's revocation system. The attack is practical because it subverts revocation checks, achieving the same outcome as a network adversary blocking CRL access, but without requiring an in-path position. Although browsers typically soft-fail on CRL retrieval [54, 86], this threat still applies to clients that perform strict revocation checks or depend on CRLs for critical functions. However, as revocation checking is being phased out in favor of short-lived certificates [8, 30, 57], this threat becomes negligible once CRLs are superseded.

5.3 Summary

TLS libraries often deviate from ASN.1 standards in Unicert parsing, likely due to interpretation differences or simplified processing. We identified issues in attribute decoding and character checks, some of which can cause security vulnerabilities like hostname validation bypass and subfield forgery.

6 Empirical Analysis of Threats

Given the entangled practices of both Unicert issuance and parsing, we explored their real-world impact across various application scenarios. We uncovered three novel threat cases: CT monitor misleading, traffic obfuscation, and user spoofing. As the scenario of user spoofing involves high attacking effort and low impact, we detail it in Appendix F.1.

6.1 CT Monitor Misleading

CT enhances PKI trust by making certificate issuance transparent [56]. Among CT components, monitors index CT logs and enable domain owners or other auditors to query certificate fields[6] (e.g., by Subject CN) to detect certificate misissuance or forgery. Prior studies show that third-party CT monitors often miss certificates [58, 91]. Thus, we suspect Unicerts with special characters may (i) hinder correct parsing and indexing by monitors and (ii) cause incorrect or incomplete query results.

Threat model. We propose a threat model, *misleading CT monitors*, where the adversary is either a malicious CA or an entity that compromises a flawed CA's infrastructure. The adversary's goal is to issue unauthorized certificates for a target domain while making them difficult for the domain owner or other auditors to detect, even though the CA's automated systems correctly log them in CT.

The attack relies on crafting certificates with special characters (e.g., NUL bytes, non-printable characters) that interfere with how CT monitors parse, index, or search certificate fields. For example, a forged certificate for a domain name may be crafted to prevent it from appearing in the monitor's search results for that domain name or its CN field. This allows the adversary to conceal their malicious activity and deceive the certificate-field-based monitoring.

[6]For example, searching CT logs via Crt.sh: https://crt.sh/?a=1

Table 6: Testing results of Unicert tolerance among CT monitors.

Monitors[1]	Query fields	Query						Present
		Case sensitive	Unicode search	Fuzzy search	U-Label check for IDN	Punycode IDN	Punycode IDN-ccTLD	Fail to return certs with special Unicode
Crt.sh	Subject (CN, O, OU, emailAddress) +SAN (DNSName, IPAddress)	×	×	✓	×	✓	✓	×
SSLMate Spotter	CN+SAN (DNSName, IPAddress)	×	×	×	✓	✓	✓	✓
Facebook Monitor	CN+SAN (DNSName)	×	×	×	✓	✓	✓[3]	×
Entrust Search[2]	CN+SAN (DNSName)	×[3]	×	×	×	✓	×	-
MerkleMap	CN+SAN (DNSName)	×	×	✓	×	✓	✓	×

[1] We only selected the monitors that offer free, public field-based query services and are accessible to us.
[2] Entrust discontinued its CT search service for trusted certificates in January 2025.
[3] The latest test results are different from those in previous work [91].

This model holds regardless of the CA's relationship with the domain owner: whether the attacker is a malicious, unauthorized CA or a compromised, authorized one, the result is the same, i.e., exploiting monitor weaknesses to conceal misissued certificates. By doing so, the adversary undermines CT's core guarantee of transparency, which is intended to enable domain owners to detect unauthorized issuance.

Problem analysis. We tested 5 public CT monitors (shown in Table 6) for key functionalities like input handling (e.g., case sensitivity, fuzzy search), support for searching Unicode strings, IDN handling, and issues about certificate presentation. The experimental settings are detailed in Appendix F.2. We observed that the public monitors show differences in certificate attribute handling, and some struggle to detect malformed Unicerts:

[P1.1] *Case-insensitive searching is a common practice.* All CT monitors can handle queries in a case-insensitive manner when querying certificates. This is secure since it allows users to retrieve all certificates related to their domain names, including detecting forged ones with case variations.

[P1.2] *Lacking fuzzy search may miss forged certificates with slight variants.* We found Entrust Search, SSLMate Spotter, and Facebook Monitor lack fuzzy search, requiring exact field inputs for successful queries. This, along with Unicode restrictions in search inputs, hinders the detection of forged certificates with minor variations, such as adding extra whitespace or multilingual characters in the CN or O fields.

[P1.3] *Failing to verify IDN legality increases the risk of undetected misuse of deceptive certificates.* All monitors support querying Punycode IDNs (A-labels) and checking formats, but Entrust Search, Crt.sh, and MerkleMap do not check special characters in domains with Unicode formats (U-labels). For instance, they accept domains with labels like "xn--www-hn0a" ("\u200ewww" in Unicode), unlike SSLMate Spotter and Facebook Monitor, which perform checks and refuse the queries. This gap allows attackers to use deceptive domains with control or invisible characters (e.g., zero-width spaces) and apply certificates for phishing or malicious use. It undermines the monitors' reliability in detecting deceptive certificates, potentially providing viability for evading traffic detection.

[P1.4] *Unicode characters can disrupt monitors' parsing and visualizing Unicerts, causing incomplete query results.* For example, SSLMate Spotter matches only the substring before "/" in the CN or ignores the CN entirely if it contains a space.

6.2 Traffic Obfuscating

Most network detection components, including firewalls, intrusion detection systems (IDS), intrusion prevention systems (IPS), and event management systems (SIEM), rely on whitelists/blacklists, rulesets, or traffic flow anomalies to identify suspicious traffic [1]. However, we found that some Unicode variations (as shown in Table 3) in certificate fields can hinder detection by these components.

Threat model. This scenario matches a TLS traffic-obfuscation threat model in which an in-path network attacker evades inspection by exploiting flaws in how detection systems parse or compare specific certificate fields. The attacker intercepts TLS (e.g., TLS 1.2 or older) handshakes and supplies a certificate crafted to trigger parsing mismatches, for example, inserting a NUL byte or extra whitespace into a Subject CN ("Evil\x00 Entity" or "Evil Entity␣"). If the detection system (e.g., firewall or IDS) relies on naive string comparisons to block or allow specific certificate fields (e.g., "CN=Evil Entity"), it will misrecognize the certificate and fail to detect the malicious entity, or corrupt logging systems to block log-based analysis [12, 69]. This may facilitate attacker traffic in evading detection and passing undetected.

Problem analysis. We tested three open-source middlebox engines (Snort [87], Suricata [72], and Zeek [101]) and four client tools (libcurl, urllib3, requests, and HttpClient) to assess their handling of field encoding and character checks in certificates (see Appendix F.2 for experiment settings). The issues we discovered include:

[P2.1] *Certificate entity parsing varies across network detection tools.* They may identify peer entities using CN, O, and SAN fields in certificates through customized parsing methods built on TLS library APIs. However, they have trouble in handling complex scenarios. For instance, Snort [92] takes the first CN/OU from duplicated Subject fields, whereas Zeek [60, 79] uses the last CN and ignores SANs not encoded as IA5String. As a result, network adversaries could evade traffic detection by crafting certificates with malicious entity information in specific positions or using non-IA5String encodings. Moreover, Suricata's case-sensitive Subject matching can be bypassed using character variants in Unicerts [16].

[P2.2] *Some client implementations lack strict SAN format checks.* Ideally, CAs and relying parties should verify SAN formats per standards, converting IDNs and IRIs to Punycode for non-ASCII domain names and avoiding conversion errors [63]. However, HTTP client implementations vary in their format checking rules. For example, urllib3 over-tolerantly restricts SAN fields to Latin-1 without checking whether IDNs are valid Punycode. In this case, a non-compliant certificate containing U-labels in SAN DNSNames could pass the client validation.

7 Discussion

Responsible disclosure. We responsibly reported the identified issues to affected CAs and TLS library teams through Email, Mozilla Bugzilla, HackerOne, and GitHub, and coordinated with them on remediation. As of September 15, 2025, we received responses or acknowledgments from 12 CAs and 6 TLS library maintainers. The disclosure timeline and outcomes are summarized in Table 7, Appendix A. For IDN validation in certificate issuance, experts from Let's Encrypt, the CA/B Forum, and Mozilla acknowledged potential risks but noted the cases do not violate current Baseline Requirements (BRs), highlighting the need to revisit BR alignment with modern IDNA [67]. Several CAs also emphasized the importance of external pre-issuance validation. For decoding and parsing issues in TLS libraries, developers and maintainers confirmed inconsistencies in handling ASN.1 and character encodings. Most teams classified these as implementation bugs rather than security vulnerabilities, unless a CA could be shown to issue technically invalid or noncompliant certificates.

Key takeaways. We summarize the following key takeaways from our discussions with stakeholders:

(1) *Achieving an internationalized PKI requires coordination across CAs, developers, and standards bodies.* While groups like ICANN and UASG promote IDNs and related web technologies [38, 44, 97], they have not explicitly addressed the complexities of Unicode integration in PKI. This gap is particularly evident with IDNs in Unicerts, where implementation is fragmented even though Punycode can bridge Unicode domain names with the ASCII-only DNS. This highlights a persistent challenge in securely and consistently handling Unicerts, from CA issuance to TLS client validation.

(2) *Integrating Unicode into certificates spans multiple evolving standards, and current CA/B BRs may lag behind some recent updates.* Our analysis of cross-referenced and multi-generational specifications indicates that the BRs, though widely followed by CAs and TLS developers, represent only a minimal consensus on technical and security standards. In our Bugzilla discussions, experts agreed that while current BRs allow IDNs conforming to Punycode syntax and being resolvable in DNS, these domains may still violate modern IDNA specifications if they include disallowed code points [31, 43], making them neither meaningful nor displayable in user agents.

(3) *Parsing ASN.1 strings in Unicerts is non-trivial.* Limited understanding and diverse interpretations of standards frequently result in absent or inconsistent checks, thereby increasing security risks and usability issues within TLS implementations. Moreover, the inherent compromises between usability and security can also lead to encoding-decoding inconsistencies, making coordinated remediation efforts by both certificate issuers and parsers particularly challenging.

Recommendations. For Unicert issuance, we propose three suggestions: (1) Prompting automated issuance strategies for TLS certificates while restricting customizable fields and character sets, as Let's Encrypt, Cloudflare, and Amazon perform (e.g., supporting customizing `DNSName` only and excluding non-domain characters), to reduce attack surfaces and complexity for both issuers and parsers. (2) Clarifying updated constraints on Unicode encoding and character checks according to the rules extracted from cross-referenced and evolving standards, including RFC 5280 and its updates, modern IDNA (IDNA2003 and IDNA2008) restrictions on names, and string preparation standards (e.g., RFC 3454 and RFC 3491). LLMs such as RFCGPT could help streamline the extraction of complex rules from standards. (3) Leveraging comprehensive certificate checking linters to ensure issuance compliance. Additionally, our extended linter can be released to the community, with plans to incorporate more rules.

For Unicert usage, we propose two suggestions: (1) The most effective solution to ASN.1 encoding-decoding mismatches is compliance with standards [46, 47] during decoding. When adopting modified decoding to handle invalid byte sequences for compatibility, developers should explicitly document the approach. (2) When parsing certificate fields into X.509-text representations, it is recommended to utilize proper data structures, such as those in Golang Crypto, to store extracted values, thereby preventing character escaping issues. If a field is parsed into a single string, any additional characters introduced (e.g., "=", "+", etc.) should be properly escaped to ensure correctness. In addition, our testing tools will be released for the community[7].

8 Conclusion

This study examined Unicert issuance and parsing compliance in the internationalized PKI, revealing challenges in Unicode adoption through CT log analysis and TLS library evaluation. From the issuance aspect, we noticed over 300 CA organizations showed noncompliance, with weak checks for illegal characters, missing normalization, and inadequate validation of critical entity information in certificate fields. Despite PKI having made notable strides in supporting Unicode, challenges like IDN validation and encoding handling remain. From the parsing aspect, the implementations of attribute decoding and character range checking for Unicerts are problematic. The issues include incompatible or over-tolerant decoding and violations in handling illegal characters. These parsing anomalies, combined with noncompliant Unicert issuance, pose security and usability risks, including user spoofing, CT monitor misleading, and traffic obfuscation, which require coordinated actions from all stakeholders.

Acknowledgments

We thank the anonymous reviewers and our shepherd for their valuable feedback, which help improve this paper. We are also grateful to our industry partner, QiAnXin, for providing dataset support, and to Shujun Tang, Zaifeng Zhang, Pengfei Liao, and Fan Yang for their assistance with this work. This research is supported by Zhongguancun Laboratory and the National Natural Science Foundation of China (62302258, 62502474). Haixin Duan is supported by the Taishan Scholars Program, and Xiang Li is supported by the National Natural Science Foundation of China (62502236). Yiming Zhang and Shenglin Zhang are both the corresponding authors.

References

[1] abuse.ch. [n. d.]. SSL Blacklist. https://sslbl.abuse.ch/.

[7]https://zeriny.github.io/unicert

[2] Mark Alllman and Vern Paxson. 2007. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement* (San Diego, California, USA) *(IMC '07)*. Association for Computing Machinery, New York, NY, USA, 135–140. doi:10.1145/1298306.1298327

[3] Arch Linux Forum. 2024. Delete Key Bug: Inputs U+007F and U+0621 in pt_BR Locale. https://bbs.archlinux.org/viewtopic.php?id=298417.

[4] Alessandro Barenghi, Nicholas Mainardi, and Gerardo Pelosi. 2018. Systematic parsing of X.509: Eradicating security issues with a parse tree. *Journal of Computer Security* 26, 6 (2018), 817–849. arXiv:https://doi.org/10.3233/JCS-171110 doi:10.3233/JCS-171110

[5] Milan Bednar. 2021. OpenVPN - malformed log - certificate subject. https://forum.netgate.com/topic/163362/openvpn-malformed-log-certificate-subject.

[6] Peter Bowen. 2016. certlint. https://github.com/amazon-archives/certlint.

[7] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. 2014. Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 114–129. doi:10.1109/SP.2014.15

[8] CA/Browser Forum. 2023. Ballot SC063v4: Make OCSP Optional, Require CRLs, and Incentivize Automation. https://cabforum.org/2023/07/14/ballot-sc063v4-make-ocsp-optional-require-crls-and-incentivize-automation/.

[9] CA/Browser Forum. 2024. Baseline Requirements for the Issuance and Management of Publicly-Trusted TLS Server Certificates, Version 2.0.7. https://cabforum.org/working-groups/server/baseline-requirements/documents/CA-Browser-Forum-TLS-BR-2.0.7.pdf.

[10] Carnegie Mellon University-CERT Coordination Center. 2022. OpenSSL 3.0.0 to 3.0.6 decodes some punycode email addresses in X.509 certificates improperly. https://kb.cert.org/vuls/id/794340.

[11] Certificate Transparency Policy. 2018. Upcoming CT Log Removal: WoSign. https://groups.google.com/a/chromium.org/g/ct-policy/c/UcCqlxuz_1c/m/Mf_939xYAQAJ?pli=1.

[12] Check Point Advisories. 2015. Multiple Vendors TLS Certificate Common Name NULL Byte Input Validation Error (CVE-2015-3008; CVE-2015-3455). https://advisories.checkpoint.com/defense/advisories/public/2015/cpai-2015-0589.html.

[13] Chu Chen, Pinghong Ren, Zhenhua Duan, Cong Tian, Xu Lu, and Bin Yu. 2023. SBDT: Search-Based Differential Testing of Certificate Parsers in SSL/TLS Implementations. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 967–979. doi:10.1145/3597926.3598110

[14] Chu Chen, Cong Tian, Zhenhua Duan, and Liang Zhao. 2018. RFC-directed differential testing of certificate validation in SSL/TLS implementations. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 859–870. doi:10.1145/3180155.3180226

[15] Chrome. 2024. Certificate Transparency Log Policy. https://googlechrome.github.io/CertificateTransparency/log_policy.html.

[16] Suricata community. [n. d.]. Suricata Rules. https://github.com/OISF/suricata/blob/aeb200e001f56982115bb8bc908a15a49373f9ec/doc/userguide/rules/differences-from-snort.rst.

[17] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and W. Timothy Polk. 2008. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *RFC* 5280 (2008), 1–151. doi:10.17487/RFC5280

[18] Adam M. Costello. 2003. Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA). RFC 3492. doi:10.17487/RFC3492

[19] curl. [n. d.]. curl. https://github.com/curl/curl.

[20] curl. 2023. CVE-2023-28321. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-28321.

[21] Unicode Character Database. 2024. Unicode Blocks. https://www.unicode.org/Public/UCD/latest/ucd/Blocks.txt.

[22] Joyanta Debnath, Sze Yiu Chau, and Omar Chowdhury. 2021. On Re-engineering the X.509 PKI with Executable Specification for Better Implementation Guarantees. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1388–1404. doi:10.1145/3460120.3484793

[23] Joyanta Debnath, Christa Jenkins, Yuteng Sun, Sze Yiu Chau, and Omar Chowdhury. 2024. ARMOR: A Formally Verified Implementation of X.509 Certificate Chain Validation . In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 1462–1480. doi:10.1109/SP54263.2024.00220

[24] Antoine Delignat-Lavaud, Martín Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, and Yinglian Xie. 2014. Web PKI: Closing the Gap between Guidelines and Practices. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society. https://www.ndss-symposium.org/ndss2014/web-pki-closing-gap-between-guidelines-and-practices

[25] Hongying Dong, Hao Shu, Vijay Prakash, Yizhe Zhang, Muhammad Talha Paracha, David Choffnes, Santiago Torres-Arias, Danny Yuxing Huang, and Yixin Sun. 2023. Behind the Scenes: Uncovering TLS and Server Certificate Practice of IoT Device Vendors in the Wild. In *Proceedings of the 2023 ACM on Internet Measurement Conference* (Montreal QC, Canada) *(IMC '23)*. Association for Computing Machinery, New York, NY, USA, 457–477. doi:10.1145/3618257.3624815

[26] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. 2013. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (Barcelona, Spain) *(IMC '13)*. Association for Computing Machinery, New York, NY, USA, 291–304. doi:10.1145/2504730.2504755

[27] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael D. Bailey, J. Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/security-impact-https-interception/

[28] Patrik Fältström. 2022. Internationalized Domain Names for Applications 2008 (IDNA2008) and Unicode 12.0.0. *RFC* 9233 (2022), 1–26. doi:10.17487/RFC9233

[29] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated Repair of Programs from Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1469–1481. doi:10.1109/ICSE48619.2023.00128

[30] Feisty Duck. 2025. Certificate Lifetimes to Shrink to Just Forty-Seven Days. https://www.feistyduck.com/newsletter/issue_124_certificate_lifetimes_to_shrink_to_just_forty_seven_days.

[31] Patrik Fältström. 2010. The Unicode Code Points and Internationalized Domain Names for Applications (IDNA). RFC 5892. doi:10.17487/RFC5892

[32] Globalsign. 2016. certlint. https://github.com/globalsign/certlint.

[33] Google. 2024. Certificate Transparency Known Logs. https://certificate.transparency.dev/google/.

[34] Google Security Blog. 2024. Sustaining Digital Certificate Security - Entrust Certificate Distrust. https://security.googleblog.com/2024/06/sustaining-digital-certificate-security.html.

[35] Internet Security Research Group. 2017. Let's Encrypt Unicode Normalization Compliance Incident. https://groups.google.com/g/mozilla.dev.security.policy/c/nMxaxhYb_iY/m/AmjCI3_ZBwAJ.

[36] HackerOne. [n. d.]. Why You Need Responsible Disclosure and How to Get Started. https://www.hackerone.com/knowledge-center/why-you-need-responsible-disclosure-and-how-get-started.

[37] David Hasselquist, Ludvig Bolin, Emil Carlsson, Adam Hylander, Martin Larsson, Erik Voldstad, and Niklas Carlsson. 2023. Longitudinal Analysis of Wildcard Certificates in the WebPKI. In *2023 IFIP Networking Conference (IFIP Networking)*. IEEE, Barcelona, Spain, 1–9. doi:10.23919/IFIPNetworking57963.2023.10186356

[38] Paul E. Hoffman and Pete Resnick. 2010. Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008. RFC 5895. doi:10.17487/RFC5895

[39] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL landscape: a thorough analysis of the x.509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (Berlin, Germany) *(IMC '11)*. Association for Computing Machinery, New York, NY, USA, 427–444. doi:10.1145/2068816.2068856

[40] Allen D. Householder, Garret Wassermann, Arthur Manion, and Christopher King. 2020. CERT® Guide to Coordinated Vulnerability Disclosure. (Sep 2020). doi:10.1184/R1/12367340.v1

[41] Russ Housley. 2018. Internationalization Updates to RFC 5280. RFC 8399. doi:10.17487/RFC8399

[42] Russ Housley. 2024. RFC 9549: Internationalization Updates to RFC 5280. *RFC* 9549 (2024), 1–10. doi:10.17487/RFC9549

[43] IANA. 2024. IDNA Rules and Derived Property Values. https://www.iana.org/assignments/idna-tables-12.0.0/idna-tables-12.0.0.xhtml.

[44] ICANN. [n. d.]. Universal Acceptance (UA). https://www.icann.org/ua.

[45] International Telecommunication Union. latest edition. ITU-T Recommendation X.509: The Directory: Public-key and attribute certificate frameworks. https://www.itu.int/rec/T-REC-X.509/en.

[46] International Telecommunication Union. latest edition. X.680 : Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation . https://www.itu.int/rec/T-REC-X.680/en.

[47] International Telecommunication Union. latest edition. 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/en.

[48] Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, and Rahul Sharma. 2022. Jigsaw: large language models meet program synthesis. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1219–1231. doi:10.1145/3510003.3510203
[49] Dan Kaminsky, Meredith L. Patterson, and Len Sassaman. 2010. PKI Layer Cake: New Collision Attacks against the Global X.509 Infrastructure. In *Financial Cryptography and Data Security*, Radu Sion (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 289–303.
[50] Erin Kenneally and David Dittrich. 2012. The Menlo report: Ethical principles guiding information and communication technology research. *Available at SSRN 2445102* (2012).
[51] Steve Kille. 1995. A String Representation of Distinguished Names. *RFC* 1779 (1995), 1–8. doi:10.17487/RFC1779
[52] Dr. John C. Klensin. 2010. Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework. RFC 5890. doi:10.17487/RFC5890
[53] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. 2018. Tracking Certificate Misissuance in the Wild. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 785–798. doi:10.1109/SP.2018.00015
[54] James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 539–556. doi:10.1109/SP.2017.17
[55] Ben Laurie, Adam Langley, and Emilia Käsper. 2013. Certificate Transparency. *RFC* 6962 (2013), 1–27. doi:10.17487/RFC6962
[56] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. 2021. Certificate Transparency Version 2.0. RFC 9162. doi:10.17487/RFC9162
[57] Let's Encrypt. 2025. Announcing Six Day and IP Address Certificate Options in 2025. https://letsencrypt.org/2025/01/16/6-day-and-ip-certs.
[58] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. 2019. Certificate Transparency in the Wild: Exploring the Reliability of Monitors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 2505–2520. doi:10.1145/3319535.3345653
[59] libESMTP. 2010. CVE-2010-1192. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1192.
[60] Zeek Log. [n. d.]. x509.log. https://docs.zeek.org/en/master/logs/x509.html.
[61] Ziyang Luo, Can Xu, Pu Zhao, Xiubo Geng, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Augmented Large Language Models with Parametric Knowledge Guiding. arXiv:2305.04757 [cs.CL] https://arxiv.org/abs/2305.04757
[62] Moxie Marlinspike. 2009. More Tricks For Defeating SSL In Practice. https://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatSSL-SLIDES.pdf.
[63] Alexey Melnikov, Wei Chuang, and Corey Bonnell. 2024. Internationalized Email Addresses in X.509 Certificates. *RFC* 9598 (2024), 1–12. doi:10.17487/RFC9598
[64] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large Language Model guided Protocol Fuzzing. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/large-language-model-guided-protocol-fuzzing/
[65] Paul V. Mockapetris. 1987. Domain names - concepts and facilities. *RFC* 1034 (1987), 1–55. doi:10.17487/RFC1034
[66] Mozilla. [n. d.]. Common CA Database (CCADB). https://www.ccadb.org/.
[67] Mozilla Bugzilla. 2025. Let's Encrypt: Issuance for Invalid Internationalized Domain Name. https://bugzilla.mozilla.org/show_bug.cgi?id=1966515.
[68] MyF5. 2019. K81239824: The X509 iRules commands may incorrectly parse SSL certificate attributes. https://my.f5.com/manage/s/article/K81239824.
[69] Netgate. 2021. OpenVPN-malformed log-certificate subject. https://forum.netgate.com/topic/163362/openvpn-malformed-log-certificate-subject.
[70] Node.js. 2021. CVE-2021-44533. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44533.
[71] Edward Oakes, Jeffery Kline, Aaron Cahn, Keith Funkhouser, and Paul Barford. 2019. A Residential Client-side Perspective on SSL Certificates. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*. 185–192. doi:10.23919/TMA.2019.8784633
[72] OISF. [n. d.]. Suricata. https://github.com/OISF/suricata.
[73] OpenSSL. [n. d.]. CVE-2022-3786 and CVE-2022-3602: X.509 Email address buffer overflows. https://openssl-library.org/post/2022-11-01-email-address-overflows/.
[74] OpenSSL. 2022. CVE-2022-3602. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-3602.
[75] OpenSSL extension in Ruby. 2015. CVE-2015-1855. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1855.
[76] OSS. [n. d.]. ASN.1 Quick Reference. https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference.html.
[77] Postfix. [n. d.]. Postfix TLS Support. http://www.postfix.org/TLS_README.html.
[78] Unicode Technical Reports. [n. d.]. Unicode Standard Annex #15: Unicode Normalization Forms", October 2006. http://www.unicode.org/reports/tr15/.
[79] Zeek repository. [n. d.]. The Zeek Network Security Monitor. https://github.com/zeek/zeek/blob/c04e503c92faa3872ed3419c2ec0327d70b62f0c/src/file_analysis/analyzer/x509/X509.cc.
[80] Kurt Roeckx. 2016. x509lint. https://github.com/kroeckx/x509lint.
[81] Jonathan Rudenberg. 2017. Certificates with invalidly long serial numbers. https://groups.google.com/g/mozilla.dev.security.policy/c/b33_4CyJbWI.
[82] Jonathan Rudenberg. 2017. Certificates with metadata-only subject fields. https://groups.google.com/g/mozilla.dev.security.policy/c/Sae5lpT02Ng.
[83] s2n-tls. 2023. Issue with parsing Certificate Common Name (CN) in s2n-tls. https://github.com/aws/s2n-tls/security/advisories/GHSA-h5p4-28rh-q272.
[84] Peter Saint-Andre and Rich Salz. 2023. Service Identity in TLS. *RFC* 9525 (2023), 1–25. doi:10.17487/RFC9525
[85] Suphannee Sivakorn, George Argyros, Kexin Pei, Angelos D. Keromytis, and Suman Jana. 2017. HVLearn: Automated Black-Box Analysis of Hostname Verification in SSL/TLS Implementations. In *2017 IEEE Symposium on Security and Privacy (SP)*. 521–538. doi:10.1109/SP.2017.46
[86] Trevor Smith, Luke Dickenson, and Kent E. Seamons. 2020. Let's Revoke: Scalable Global Certificate Revocation. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/lets-revoke-scalable-global-certificate-revocation/
[87] Snort 3.0 Team. [n. d.]. Snort++. https://github.com/snort3/snort3.
[88] Rob Stradling. 2023. pkimetal. https://github.com/pkimetal/pkimetal.
[89] Nick Sullivan. 2024. LLMS and RFCGPT - Leveraging large language model platforms to understand standards. https://datatracker.ietf.org/meeting/119/materials/slides-119-rasprg-llms-and-rfcgpt-leveraging-large-language-model-platforms-to-understand-standards-01.
[90] Nick Sullivan. 2024. RFCGPT. https://cryptography.consulting/rfcgpt.
[91] Aozhuo Sun, Jingqiang Lin, Wei Wang, Zeyan Liu, Bingyu Li, Shushang Wen, Qiongxiao Wang, and Fengjun Li. 2024. Certificate Transparency Revisited: The Public Inspections on Third-party Monitors. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/certificate-transparency-revisited-the-public-inspections-on-third-party-monitors/
[92] Snort 3.0 Team. [n. d.]. snort3 ssl.cc. https://github.com/snort3/snort3/blob/8f8e9cf28856359a1ef3081baa9240f90276f8d4/src/protocols/ssl.cc.
[93] The Chromium Projects. [n. d.]. Moving Forward, Together. https://www.chromium.org/Home/chromium-security/root-ca-policy/moving-forward-together/.
[94] The Register. 2025. Bug hunter tricked SSL.com into issuing cert for Alibaba Cloud domain in 5 steps. https://www.theregister.com/2025/04/22/ssl_com_validation_flaw/.
[95] The Unicode Consortium. [n. d.]. Unicode 16.0 Character Code Charts. https://unicode.org/charts/PDF/U0000.pdf.
[96] The ZMap Project. 2016. ZLint. https://github.com/zmap/zlint/.
[97] UASG. 2024. UASG 050 UA-Readiness Report FY24. https://uasg.tech/download/uasg-050-ua-readiness-report-fy24/.
[98] Mark Wahl, Steve Kille, and Tim Howes. 1997. Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. *RFC* 2253 (1997), 1–10. doi:10.17487/RFC2253
[99] Jincheng Wang, Le Yu, and Xiapu Luo. 2024. LLMIF: Augmented Large Language Model for Fuzzing IoT Devices. In *2024 IEEE Symposium on Security and Privacy (SP)*. 881–896. doi:10.1109/SP54263.2024.00211
[100] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. 2020. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Comput. Surv.* 53, 3, Article 63 (June 2020), 34 pages. doi:10.1145/3386252
[101] Zeek Network Monitoring Project. [n. d.]. The Zeek Network Security Monitor. https://github.com/zeek/zeek.
[102] Kurt D. Zeilenga. 2006. Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names. *RFC* 4514 (2006), 1–15. doi:10.17487/RFC4514

A Ethics

Ethical considerations. We adhered to ethical standards outlined in the Menlo Report [50] and guidelines for public dataset use [2]. Our analysis of CA issuance practices relied on publicly available CT log data, which is widely used in PKI research [58, 91], posing no ethical concerns. The assessments of TLS libraries and threat surfaces were conducted in controlled experimental environments.

Table 7: Timeline and outcomes for responsible disclosure.

TLS Libraries	Timeline	Outcomes
PyOpenSSL	2024.05.06: Reported Unicode handling inconsistencies. 2025.10.26: Received a response	Claimed their X509 APIs are deprecated with the intent to remove them; recommended using the X.509 APIs from pyca/cryptography instead.
	2025.02.11: Reported another issue. 2025.02.18 & 03.17: Follow-up reports.	Issue closed on GitHub without a response.
Golang Crypto	2025.02.11: Initial report. 2025.02.20: Received a response.	Confirmed and fixed the issue (Go issue #72078).
OpenSSL	2025.02.11: Initial report. 2025.03.17: Follow-up report. 2025.04.04: Received a response.	Acknowledged inconsistencies across similar functions; stated affected functions could be deprecated.
Node.js Forge	2025.02.11: Initial report. 2025.03.11: Follow-up report.	No response received.
Node.js Crypto	2025.02.11: Initial report. 2025.05.19: Follow-up report. 2025.05.21 - 2025.07.04: Interactive communication.	Confirmed issue as a bug, but not a vulnerability if CAs do not issue invalid certificates.
GnuTLS	2025.02.11: Initial report. 2025.03.26 & 04.26: Follow-up reports.	No response received.
BouncyCastle	2025.02.11: Initial report. 2025.02.17: Additional documents provided.	No response received.
Cryptography	2025.02.13: Initial report. 2025.02.17: Received a response.	Confirmed issues; explained lax handling of certain ASN.1 string types for compatibility; advised avoiding deprecated APIs.
Java.security.cert	2025.03.03 - 2025.07.18: Interactive communication.	Confirmed issues; committed to addressing them in a future Critical Patch Update.

CA Entities	Timeline	Outcomes
Let's Encrypt	2025.05.14: Initial report. 2025.05.15: Received a response.	Acknowledged potential security risks, discussed via Bugzilla (Bug #1966515), but stated no violation of current CA/B BRs.
Sectigo	2025.05.13: Initial report. 2025.05.13 & 14: Received responses.	Acknowledged receipt; no further response.
DigiCert	2025.06.24: Initial report. 2025.06.27 & 07.09 & 08.04: Follow-up reports.	Acknowledged receipt; requested further investigation of other CA brands chaining to DigiCert roots; no further reply.
Netlock	2025.06.24: Initial report. 2025.06.25: Received a response.	Acknowledged issue; claimed improvements already deployed (enhanced pre-issuance validation, pkimetal checks); emphasized the importance of external validation.
Telia, PostSignum, ANF AC	2025.07.09: Initial report. 2025.07.09 & 10: Received a response.	Acknowledged known issue; stated it was already resolved.
ACCV	2025.07.09: Initial report. 2025.07.10: Received a response.	Confirmed risk/incident; stated issue was already resolved.
Certum	2025.07.09: Initial report. 2025.07.09: Received a response.	Claimed compliance with RFC 5280; stated strict validation is in place.
GlobalSign	2025.07.09: Initial report. 2025.07.11: Received a response.	Claimed encoding errors are detected pre-issuance.
Telekom	2025.07.09: Initial report. 2025.08.04: Follow-up report. 2025.08.08: Received a response.	Attributed issues to a temporary misconfiguration that has since been fixed.
Certicámara	2025.07.09: Initial report.	Confirmed issue was fixed.
AOC CAT, Camerfirma	2025.07.09: Initial report.	These CAs have ceased issuing certificates.
GoDaddy	2025.07.09: Initial report. 2025.08.04: Follow-up report.	Acknowledged receipt; no further reply.
NISZ	2025.07.09: Initial report. 2025.08.04: Follow-up report.	Acknowledged receipt; no further reply.
E-tugra, FNMT, Microsec, D-TRUST	2025.07.09: Initial report. 2025.07.11 & 21: Follow-up reports.	No response received.

This summarizes disclosure responses received up to September 15, 2025.

Responsible disclosure and timeline. Following responsible disclosure guidelines [36, 40], we notified affected vendors and developers in phases aligned with our research timeline, allowing sufficient time to implement fixes before publication. Our objective was to disclose potential security and technical issues in a manner most beneficial to the community.

We first contacted TLS library maintainers, as early experiments focused on flaws in certificate parsing and decoding. The Unicode handling bug in PyOpenSSL was reported during this initial phase, while disclosures for other TLS libraries followed. Notifications to CA issuers came later, as some analyses were completed during the revision stage of this paper. Table 7 summarizes the disclosure timeline. We also issued follow-up notifications to entities that did not respond initially, with detailed outcomes documented in the table.

B Term Introduction

This section provides a supplementary introduction to the terminology related to ASN.1 string types (Table 8), certificate fields (Table 9), and others (Table 10), with a particular focus on those used throughout this paper.

C Settings and Prompts for Standard Extraction

This section introduces details of standard analysis tasks and relevant LLM prompts for Section 3.1.1. Given a desired certificate attribute, we constructed two tasks to derive standard requirements: (i) identifying valid encoding types and data structures, and (ii) summarizing encoding and format rules.

Table 8: ASN.1 string types used in RFC 5280.

String Types	Tag	Description
UTF8String	12	This type uses a variable-width encoding(UTF-8) that supports the entire Unicode character set (ISO/IEC 10646), making it ideal for internationalization.
NumericString	18	This type uses ASCII encoding and is restricted to representing only digits (0-9) and the space character.
PrintableString	19	This type uses ASCII encoding and supports upper case letters [A-Z], lower case letters [a-z], the digits [0-9], space, and common punctuation marks. It does not support the "@", "&", and "*" characters.
IA5String	22	This type uses ASCII encoding and the corresponding charset is equivalent to the 7-bit ASCII character set (International Alphabet 5), containing 128 characters, and is often used for email addresses or DNS names.
VisibleString	26	This type uses ASCII encoding and encompasses all visible (printable) characters of the IA5 (ASCII) character set, excluding control characters.
UniversalString	28	This type uses UCS-4 encoding and represents characters from the entire Universal Character Set (UCS, ISO/IEC 10646) using four octets per character.
BMPString	30	This string type encodes characters using two octets per character(UCS-2), representing the Basic Multilingual Plane (BMP) of Unicode (U+0000 to U+FFFF), which covers most common scripts.
TeletexString	20	This string type employs T.61 encoding, which supports multiple character sets for Teletex machines. It frequently uses escape sequences to switch between these sets.

Table 9: Main certificate fields mentioned in this paper.

Certificate Fields	Description
Subject/Issuer	
CN/CommonName	The primary identifier of the certificate's subject, often a hostname for servers.
C/CountryName	The two-letter ISO country code of the certificate's subject or issuer.
L/LocalityName	The city or locality of the certificate's subject or issuer.
O/OrganizationName	The legal name of the organization associated with the certificate's subject or issuer.
Extensions	
SAN/SubjectAltName	Alternative identities for the certificate's subject, such as multiple DNS names, IP addresses, or email addresses.
IAN/IssuerAltName	Alternative identities for the certificate's issuer, similar to SAN but for the certificate authority.
CP/CertificatePolicies	The policies under which the certificate was issued and its intended use.
CRLDistributionPoints	Locations where Certificate Revocation Lists (CRLs) for the certificate can be found.
AIA/AuthorityInfoAccess	Information about how to access the issuer's certificate and/or online certificate status protocol (OCSP) services.
SIA/SubjectInfoAccess	Information about how to access information and services for the subject of the certificate.

Table 10: Other commonly used terms.

Terms	Description
Punycode [18]	An encoding syntax that represents Unicode characters in a limited ASCII character subset used for IDNs.
A-Label	The ASCII Compatible Encoding (ACE) form of an IDN, beginning with the prefix "xn--", which could be the output of Punycode algorithm.
U-Label	The Unicode form of an IDN, representing the domain name with its original, native script characters.
NFC [78]	Unicode Normalization Form C, a canonical composition form that combines characters and their combining marks into single, precomposed characters.
Attribute normalization	When the UTF8String encoding is used, all character sequences SHOULD be normalized according to NFC.

For the first task, we extracted inclusion relationships among data structures (e.g., DistinguishedName) and respective ASN.1 encoding types (e.g., UTF8String). Given specific encoding types, we instructed the LLM to identify its character ranges based on attribute types, as the same encoding type may enforce different ranges depending on the attribute. For instance, DNSName and EmailAddress can be encoded as IA5String; however, DNSName allows only [a-zA-Z0-9.-], while EmailAddress can include "@". For the second task, we had the LLM summarize relevant requirements based on the background context and attribute name.

We used the prompts constructed through the templates in Figure 5 and Figure 6. Since certain certificate attributes (e.g., CertificatePolicies, CRLDistributionPoints) have more complex structures than common attributes (e.g., CommonName), we need to repeat the queries across multiple rounds and enforce detailed instructions, such as specifying options for sub-attributes. Meanwhile, we pre-set a sliding window for the inputs and instructed the model to generate condensed summaries when the input exceeded the LLM's input limit.

Prompt Template

Instruction:
Present relationships among the following data structures and encoding types: [DistinguishedName, RDNSequence, UTF8String...]. Use '-->' to denote a relationship between them.

Requirements:
1. If a structure contains others, describe each layer in sequence.
2. Point each ASN.1 structure directly to its encoding types.
3. Point each encoding type to the character sets.

Example:
If a structure GeneralName contains DNSName, and DNSName should be serialized as IA5String, your output would be:
GeneralName-->DNSName-->IA5String (ASCII characters, excluding '@')

Desired Output Format:
Shot-1: DistinguishedName-->RDNSequence-->DirectoryString
Shot-2: GeneralName-->IPAddress (binary/octet string)

Figure 5: Prompt template for drawing a directed graph to present data structure relationships.

Prompt Template

Instruction:
You are a protocol analyst. For a given [desired cert attribute] and provided [background context], extract the attribute's permissive data structures, encoding types, and any relevant requirements on encoding and string format.

Requirements:
1. Format "structures" as a nested JSON if there are hierarchical inclusions, e.g., {"GeneralName" : {"DNSName" : {}}. Optional structures include [list optional structures].
2. For "requirements", return a list of relevant encoding or format restrictions as raw text. If no restrictions apply, respond with "No."

Cert attribute: [desired cert attribute]

Background context: [knowledge]

Desired Output Format:
Shot-1:
```
"SubjectAltName": {
    "structures": "GeneralName",
    "requirements": [
        "When the SubjectAltName extension contains a domain name system
        label, the domain name MUST be stored in the DNSName (an IA5String)", ...]
}
```

Figure 6: Prompt template for gathering encoding and format requirements.

Table 11: Top 25 lints identifying noncompliant cases.

Lint Name	Lint Type	New Lint	Level	#NC Unicerts
w_rfc_ext_cp_explicit_text_not_utf8	Invalid Encoding		SHOULD	117,471
w_cab_subject_common_name_not_in_san	Invalid Structure		MUST	93,664
e_rfc_dns_idn_a2u_unpermitted_unichar	Invalid Character	✓	MUST	26,701
e_subject_organization_not_printable_or_utf8	Invalid Encoding	✓	MUST	25,751
e_subject_common_name_not_printable_or_utf8	Invalid Encoding	✓	MUST	25,081
e_subject_locality_not_printable_or_utf8	Invalid Encoding	✓	MUST	17,825
e_rfc_subject_dn_not_printable_characters	Invalid Character		MUST	13,320
e_subject_ou_not_printable_or_utf8	Invalid Encoding	✓	MUST	11,654
e_subject_jurisdiction_locality_not_printable_or_utf8	Invalid Encoding	✓	MUST	4,213
e_rfc_ext_cp_explicit_text_too_long	Illegal Format		MUST	2,988
e_subject_jurisdiction_state_not_printable_or_utf8	Invalid Encoding	✓	MUST	2,829
e_rfc_ext_cp_explicit_text_ia5	Invalid Encoding		MUST	2,550
e_subject_jurisdiction_country_not_printable	Invalid Encoding	✓	MUST	1,744
e_subject_state_not_printable_or_utf8	Invalid Encoding	✓	MUST	1,671
e_rfc_subject_printable_string_badalpha	Invalid Character		MUST	1,561
w_community_subject_dn_trailing_whitespace	Invalid Character		SHOULD	1,356
e_subject_postal_code_not_printable_or_utf8	Invalid Encoding	✓	MUST	1,262
e_subject_street_not_printable_or_utf8	Invalid Encoding	✓	MUST	990
w_cab_subject_contain_extra_common_name	Discouraged Field		SHOULD	589
e_subject_dn_serial_number_not_printable	Invalid Encoding		MUST	461
w_community_subject_dn_leading_whitespace	Invalid Character		SHOULD	437
e_rfc_subject_country_not_printable	Invalid Encoding		MUST	409
e_rfc_dns_idn_malformed_unicode	Invalid Character		MUST	401
e_cab_dns_bad_character_in_label	Invalid Character		MUST	326
e_ext_san_dns_contain_unpermitted_unichar	Invalid Character	✓	MUST	109

[1] NC Unicerts: Non-compliant Unicerts detected by the relevant lint.

D List of Unicert Checking Lints

We present in Table 11 the lints that have detected over 100 non-compliant cases in our measurement, helping evaluate the scope and severity of the issues more effectively.

E Experiment Settings of TLS Library Analysis

This section introduces detailed experiment settings for analyzing TLS libraries in Section 3.2 and Section 5.

The tested TLS library and API lists. The functions tested in our analysis are listed in Table 12 and Table 13, with some being class methods rather than standalone functions.

Experiment settings for test certificate generation. The test certificates cover the following parameters:

- OIDs of the certificate attribute types: 2.5.4.3, 2.5.4.5, 2.5.4.7, 2.5.4.8, 2.5.4.10, 2.5.4.11, 2.5.4.15, 0.9.2342.19200300.100.1.25, and 1.2.840.113549.1.9.1.
- ASN.1 encoding types: PrintableString, UTF8String, IA5String, and BMPString.
- GeneralName types: DNSName, RFC822Name, and URI.

Details of character checking in TLS libraries. In analyzing character checking errors (Section 5.2), we excluded the following cases: (i) fields parsed into structured data (e.g., Go Crypto's DN parsing) rather than an X.509-text string, where escaping checks were not applicable; (ii) fields parsed into an X.509-text string with an explicitly stated RFC (e.g., Cryptography's DN parsing per RFC4514), where other RFCs were not assessed; (iii) libraries unable to parse all extensions (e.g., BouncyCastle); and (iv) instances of incompatible decoding, where misidentified Unicode characters made character handling irrelevant.

F Additional Threat Scenario

F.1 Scenario: User Spoofing

Beyond the threat scenarios in Section 6, we identified another novel scenario where adversaries might trick browsers with malformed Unicerts to facilitate user spoofing attacks. Given its high attack conditions and limited immediate impact, we include the discussion in the appendix to invite further exploration from interested researchers and stakeholders.

Problem introduction. Visualizing X.509 certificates in browsers helps users assess website entity information, with key fields (e.g., issuer, subject) rendered as string representations in components like certificate boxes in address bars. However, errors in Unicode handling and rendering in Unicerts can mislead users. While some browsers have upgraded IDN display policies, many still fail to adequately inspect IDNs and non-ASCII characters in Unicerts, facilitating spoofing attacks. This section evaluates browser certificate rendering components by analyzing visual implementations in popular browsers and their susceptibility to spoofing users with abnormal Unicerts.

Problem analysis. We tested popular browsers using malformed Unicerts with special Unicode, such as combining and formatting characters, focusing on how certificate components render Unicode attributes. We examined whether browsers visually display special Unicode to help users identify issues, validate character ranges, and handle deceptive characters. We also assessed the effectiveness of warning pages in helping users understand connection failures. Table 14 summarizes our findings, highlighting differences in browser practices:

[G1.1] *Browsers' implementations vary in visualizing control characters.* Most browsers mark non-printable C0/C1 codes with visual indicators like Unicode symbols or URL encoding (e.g., %00), while only Firefox uses robust but potentially insecure rendering. Instead, invisible layout codes (U+2000~U+206F) are invisible across

Table 12: Tested TLS libraries and APIs for loading certificates and parsing Subject/Issuer fields.

TLS Libs	Version	Subject	Issuer	LoadCert

OpenSSL	**3.3.0**	X509_NAME_oneline()
		X509_NAME_print()
		X509_NAME_print_ex()
			X509_NAME_oneline()
			X509_NAME_print()
			X509_NAME_print_ex()
				PEM_read_bio_X509()

GnuTLS	**3.7.11**	gnutls_x509_crt_get_subject_dn()
		gnutls_x509_crt_get_subject_dn3()
			gnutls_x509_crt_get_issuer_dn()
			gnutls_x509_crt_get_issuer_dn3()
				gnutls_x509_crt_import()

PyOpenSSL	**24.2.1**	get_subject()	get_issuer()	load_certificate()
CryptoGraphy	**42.0.7**	subject.rfc4514_string()	issuer.rfc4514_string()	load_der_x509_certificate()
Go Crypto	**1.23.0**	Subject.ShortName	Issuer.ShortName	ParseCertificate()

Java security.cert	**1.8/11.0/17.0/21.0**	getSubjectDN().toString()
		getSubjectDN().getName()
		getSubjectX500Principal().getName()
		getSubjectX500Principal().toString()
			getIssuerDN().toString()
			getIssuerDN().getName()
			getIssuerX500Principal().getName()
			getIssuerX500Principal().toString()
				CertificateFactory.getInstance("X.509")
				.generateCertificate()

BouncyCastle	**1.78.1**	getSubject().toString()	getIssuer().toString()	X509CertificateHolder()
Forge	**1.3.1**	subject.getField()	issuer.getField()	X509Certificate()
Node.js Crypto	**22.4.1**	subject	issuer	certificateFromPem()

[1] Some APIs in the table have multiple parameter settings, which we do not showcase here.
[2] After importing the certificate, there are two ways to obtain the parsed values: by calling functions or by accessing attributes.

Table 13: Tested TLS libraries and APIs for parsing certificate extensions.

TLS Libs	SAN	IAN	AIA	CRLs	SIA
OpenSSL	-	-	-	-	-
GnuTLS	gnutls_x509_crt_get_subject_alt_name()	gnutls_x509_crt_get_issuer_alt_name()	-	gnutls_x509_crt_get_crl_dist_points()	-
PyOpenSSL	str(get_extension())				-
CryptoGraphy	get_extension_for_oid().value				
Go Crypto	SubjectAlternativeName	-	-	CRLDistributionPoints	-
Java security.cert	getSubjectAlternativeNames()	getIssuerAlternativeNames()	-	-	-
BouncyCastle	-	-	-	-	-
Forge	getExtension()		-	-	-
Node.js Crypto	subjectAltName	-	infoAccess	-	-

- The TLS library cannot parse this field.

Table 14: Certificate visualization and potential spoofing issues in mainstream browsers.

Browser	Version	Kernel	Components	Checking and Rendering Results					Warning Pages
				C0 & C1 controls	**Layout controls**	**Homograph feasibility**	**Incorrect substitutions**	**Flawed ASN.1 range checking**	**Spoofing feasibility**
Firefox	v. 141.0	Gecko	Digest/Details	∅	∅	✓	✓	✓	✓
			General	-	-	-	-	-	
Safari	v. 17.6	Webkit	Digest/Details	●	∅	✓	✓	✓	×
			General	-	-	-	-	-	
Chromium-based[1]	/	Blink	All parts	●	∅	✓	✓	×	✓

[1] We tested several Chromium-based browsers, including Chrome (v. 139.0), Edge (v. 128.0), Brave (v. 1.73.97), Opera (v. 114.0), Yandex (v. 24.12.1), and 360 Browser (v. 14.1). Despite their custom implementations, their certificate rendering modules function similarly, allowing us to consolidate their results.
[2] ∅: Invisible; ●: Visible; ✓: The component is vulnerable;×: The component is not vulnerable; -: no such components.

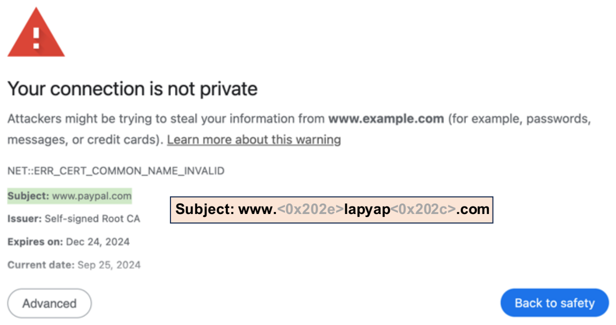
browsers, enabling attackers to craft deceptive Unicerts, leaving users unable to spot threats via certificate rendering.

[G1.2] *Detecting character similarities in certificate components is problematic.* Browsers fail to detect visual similarities (e.g., Cyrillic-Latin homographs) in UTF8Strings within Unicerts, making it feasible to facilitate homograph attacks. They also misapply equivalent character substitution policies, such as converting the Greek question mark (U+037E) to a semicolon (U+003B) instead of the correct Latin question mark (U+003F), violating Unicode standards [95].

[G1.3] *Crafting certificate fields can manipulate browser warning pages.* Browsers generate warning pages using server certificate details. For example, Chromium-based browsers (e.g., Chrome, Edge) prioritize subject fields (CN, O, OU) with UTF8String support, while Firefox uses SAN DNSNames. These details in warning pages could guide users on whether to proceed with the "insecure" website access. However, we found that these warning pages can render control characters, allowing manipulation via crafted certificate fields. Here, we presented two examples of generating spoofing warning pages using crafted certificates.

First, we found that inserting bidirectional control characters into CN fields can cause all tested Chromium-based browsers to display "www.\u202elapyap\u202c.com" as "www.paypal.com", as the Chrome example shown in Figure 7.

Second, we found customized SANs can alter Firefox warnings. We provided Firefox with a certificate containing a Subject CN with a long descriptive string ("port 8443. But they're the same site..."). In this case, the browser generates the alert information based on the crafted certificate fields (Figure 8), which can mislead users and reduce their vigilance.



Your connection is not private

Attackers might be trying to steal your information from **www.example.com** (for example, passwords, messages, or credit cards). Learn more about this warning

NET::ERR_CERT_COMMON_NAME_INVALID

Subject: www.paypal.com

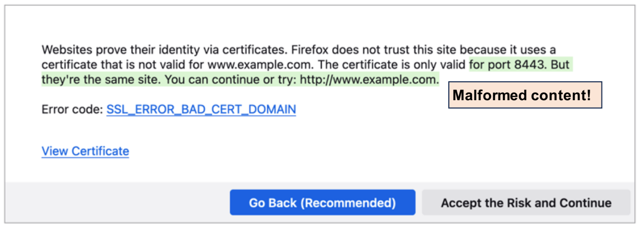
Issuer: Self-signed Root CA

Expires on: Dec 24, 2024

Current date: Sep 25, 2024

Subject: www.<0x202e>lapyap<0x202c>.com

Figure 7: A spoofing warning page in Chrome.



Websites prove their identity via certificates. Firefox does not trust this site because it uses a certificate that is not valid for www.example.com. The certificate is only valid for port 8443. But they're the same site. You can continue or try: http://www.example.com.

Malformed content!

Error code: SSL_ERROR_BAD_CERT_DOMAIN

Figure 8: A spoofing warning page in Firefox.

F.2 Experimental Setup

This section introduces detailed experiment settings that are necessary for understanding the threat analysis in each scenario in Section 6 and Section F.1.

Experiment settings for CT monitor misleading analysis. We focused on open CT monitors that offer public, free services, including Entrust Search, Crt.sh, SSLMate Spotter, Facebook Monitor, and MerkleMap. These monitors allow users to search certificates by SAN and a limited subset of Subject attributes, as shown in Table 6. To test their search and display functionality, we sampled 1 thousand noncompliant Unicerts, especially selecting those containing non-printable characters in the CN, O, OU, and SAN fields. We evaluated their handling of Unicerts with invalid characters, illegal formats, and flawed normalization.

Experiment settings for traffic obfuscating analysis. We designed an experiment to assess the feasibility of evading network detection using Unicerts. Based on the adoption popularity, we selected the latest versions of various open-source software, including three middlebox engines (Snort [87], Suricata [72], and Zeek [101]) and four client implementations (libcurl, urllib3, requests, and HttpClient). For middleboxes, we analyzed TLS certificate usage and validation rules by reviewing their source code and documentation. For client implementations, we set up a test TLS server that provides and rotates the test certificates generated in Section 5 and conducted dynamic execution tests to observe how they parse and handle these certificates.

Experiment settings for user spoofing analysis. We tested widely used web browsers (Table 14) on Ubuntu 20.04.6, Windows 11, and macOS Sonoma 14.6.1. Most of them use the Chromium-based Blink engine, showing similar rendering behaviors. The test Unicerts were generated as described in Section 3.2, focusing on exploitable Unicode characters like combining and formatting codes. We set up an HTTPS server with these Unicerts and examined how browser components render the crafted values. We used these Unicerts to examine browser components' rendering of Unicode attributes. The tested components include *digest module* like Certificate Viewer in Chrome, a *general module* outlining key fields, a *detail module* listing all fields, and *warning pages* alerting web users for potential issues. For components with anomalies, we further tested their defense mechanisms using malformed certificates.