Checking the X.509 Certificate Linter Zlint Based on Request for Comments 5280

Pinghong Ren, Chu Chen* and Donggang Yang

School of Computer Science, Qufu Normal University, Rizhao, Shandong, 276826, China

*Corresponding author’s e-mail: rzqfnu@yeah.net

Abstract. The X.509 certificate linter Zlint is helpful to trace changes that a certificate conforms to and violates rules in a process of certificate mutation. Therefore, the correctness of Zlint is very important. We put forward an approach for checking Zlint based on RFC 5280. First, rules adopted by Zlint are extracted from Zlint and checked against rules specified in RFC 5280 to reveal discrepancies. Second, certificates violating intersected rules are generated and employed to test the program of Zlint to find out whether Zlint outputs correct analytical results. Third, rules in RFC 5280 but not in Zlint are analyzed to find out latent missing functions of Zlint. We have implemented our approach and conducted comprehensive experiments. Experimental results show that some rules adopted by Zlint are informal, some analytical results produced by Zlint are not correct, and Zlint misses some functions related to the rules of RFC 5280.

1. Introduction

Certificate validation in the Secure Sockets Layer or Transport Layer Security (SSL/TLS) protocol is very important to the security of the Internet. The correctness of certificates is a key point in certificate validation and many approaches such as Mucert [1] and mRFCcert [2] adopt certificate mutation to generate certificates to test certificate validation. The X.509 certificate linter Zlint [3] employs rules specified in specifications such as RFC 5280 [4] to analyze which rules a certificate conforms to or violates. Zlint is useful for tracing the state of a certificate conforming to and violating rules during certificate mutation. Thus, the correctness of Zlint is critical to certificate analysis and mutation.

To check the correctness of Zlint, we put forward an approach based on the specification RFC 5280. Our approach firstly checks the basis of Zlint by extracting rules from Zlint and comparing them with the rules specified in RFC 5280 to detect discrepancies between rules employed by Zlint and rules specified in RFC 5280. After that, our approach obtains the rules commonly employed or specified by Zlint and RFC 5280. These intersected rules are employed to generate certificates which violate them. The generated certificates act as test cases to check the program of Zlint whether it outputs correct analytical results. Besides the intersected rules, our approach employs rules which are not included in Zlint to analyze latent missing functions of Zlint.

Our approach named ZlintR5checker has the following advantages: (1) ZlintR5checker checks the basic rules and then the program of Zlint comprehensively; (2) ZlintR5checker analyses rules which do not exist in Zlint to find missing functions; and (3) ZlintR5checker is effective in finding issues in both rules adopted by Zlint and the program of Zlint, and revealing missing functions.
The remainder is arranged as follows. The next section introduces preliminaries which are helpful to understand ZlintR5checker. Section 3 presents our approach in detail and then experimental results are discussed in Section 4. Section 5 briefly introduces related work and Section 6 concludes this work.

2. Preliminaries

The capitalized key words that express requirement or prohibition levels, classification of rules, and structure of certificates are introduced briefly as follows.

2.1. Capitalized key words and classification of rules

RFC 2119 [5] emphasized that eleven capitalized key words should be used in Internet Engineering Task Force (IETF) documents to express different requirement or prohibition levels as follows.

1. Expressing absolute requirements or prohibitions: “MUST”, “REQUIRED”, “SHALL”, “MUST NOT”, or “SHALL NOT”;
2. Expressing flexible requirements or prohibitions: “SHOULD”, “RECOMMENDED”, “SHOULD NOT”, or “NOT RECOMMENDED”; and
3. Expressing truly optional requirements: “MAY” or “OPTIONAL”.

Sentences including these capitalized key words are called rules and the requirement or prohibition levels of key words determine the classification of rules. For example, sentences including “MUST”, “REQUIRED”, “SHALL”, “MUST NOT”, or “SHALL NOT” are rules about absolute requirements or prohibitions.

2.2. Structure of certificates

An X.509 certificate consists of three parts as follows.

1. tbsCertificate: a to-be-signed certificate contains ten fields i.e., version, serial number, signature algorithm, issuer, issuer unique identifier, validity, subject, subject unique identifier, subject public key info, and extensions;
2. signature algorithm; and
3. signature value.

The field “extensions” has 15 standard extensions, 2 private Internet extensions and other Netscape extensions. More details about 17 standard and private Internet extensions are present in RFC 5280.

3. RFC5280-based approach for checking Zlint

Figure 1 shows the overview of our approach ZlintR5checker. At the first step, ZlintR5checker extracts the basis i.e., rules from Zlint and compares them with rules extracted from RFC 5280 to reveal discrepancies. At the following step, ZlintR5checker employs rules found in both Zlint and RFC 5280 to generate certificates for testing the program of Zlint to detect issues. At the final step, rules found in RFC 5280 but not in Zlint are analysed to find out missing functions.

![Figure 1. Overview of ZlintR5checker.](image)

3.1. Revealing rule discrepancies between Zlint and RFC 5280

Based on the rules extracted from RFC 5280 by RFCcert [6], Algorithm 1 presents the method for revealing rule discrepancies between Zlint and RFC 5280.
Algorithm 1: Revealing rule discrepancies between Zlint and RFC 5280

**Input**: rules extracted from RFC 5280 (denoted by \( \text{R}_{\text{RFC5280}} \))

**Output**: rule discrepancies between Zlint and RFC 5280

1. \( I_{\text{Zlint}} = \{ \text{lint related to RFC 5280} \} \);
2. \( \text{foreach } i \in I_{\text{Zlint}} \text{ do} \)
3. \( R_{\text{Zlint}} += \{ \text{rule } | \text{ rule contained in } i \} \);
4. \( \text{end} \)
5. \( R_{Z-R} = R_{\text{Zlint}} - R_{\text{RFC5280}} \);
6. \( R_{R-Z} = R_{\text{RFC5280}} - R_{\text{Zlint}} \);
7. \( R_{ZR} = R_{\text{Zlint}} \cap R_{\text{RFC5280}} \);
8. \( \text{foreach } r \in R_{ZR} \text{ do} \)
9. \( R_{ZRI} += \{ r \mid r \text{ includes lowercase key words} \} \);
10. \( \text{end} \)

Options "--includeSources" and "--list-lints-json" provided by Zlint output desired information from some sources. With these options, a set of lint related to RFC 5280 is obtained in Line 1 of Algorithm 1. In Lines 2-4, rules are extracted from \( I_{\text{Zlint}} \) and form the set \( R_{\text{Zlint}} \). Rules existing in Zlint but not in RFC 5280 form the set \( R_{Z-R} \) in Line 5. There are three cases in \( R_{Z-R} \): (1) Rules specified in RFCs other than RFC 5280 are cited in RFC 5280; (2) Sentences without key words; and (3) Sentences has key words but the original sentences found in the appendix of RFC 5280 has no key words. In Line 6, rules missed by Zlint form the set \( R_{R-Z} \). The missing rules will be further analyzed at the third step. Rules in common form the set \( R_{ZR} \) and then are checked in Lines 7-10 whether informally lowercase key words are included. The sets \( R_{Z-R}, R_{R-Z} \), and \( R_{ZRI} \) show rule discrepancies between Zlint and RFC 5280, and the process of these rule set needs human assistance since Zlint does not cite original rules. The following step makes use of rules in common.

3.2. Employing rules in common to test Zlint

Based on the rules in common, Algorithm 2 presents the method for testing the program of Zlint.

Algorithm 2: Employing rules in common to test Zlint

**Input**: rules in common (i.e., \( R_{ZR} \))

**Output**: issues of Zlint

1. \( C = \{ \text{cert} \mid \text{cert. is generated to violate } r \in R_{ZR} \} \);
2. \( \text{foreach } \text{cert. } \in C \text{ do} \)
3. \( o_{r} = \text{Zlint(cert.);} \)
4. \( R_{r} = \{ \text{rule } | \text{ the rule is reported to be violated in } o_{r} \}; \)
5. \( D_{r} = R_{r} \oplus \{ r \}; \)
6. \( D += D_{r}; \)
7. \( \text{end} \)

In Line 1 of Algorithm 2, certificates violating rules which are found in both Zlint and RFC 5280 are generated and then employed to check whether the program of Zlint outputs correctly. In Lines 3 and 4, an output (i.e., \( o_{r} \)) is produced by Zlint for an input \( \text{cert.} \) and then a rule set \( R_{r} \) that Zlint reports being violated are extracted from the output \( o_{r} \). The discrepancies between \( R_{r} \) and the rule set \( \{ r \} \) in Lines 5 and 6 are clues to detect issues of Zlint. The next step exploits rules missed by Zlint.

3.3. Analyzing missing rules to find out missing functions

\( R_{R-Z} \) obtained in Algorithm 1 is a set of rules which are specified in RFC 5280 but do not appear in Zlint. Rules in \( R_{R-Z} \) are analysed to determine whether corresponding functions are missed by Zlint. This analysis work also need human assistance due to the difficulty of inferring functions from missing rules and evaluating the feasibility to realize such functions.
4. Experiments
We have implemented our approach ZlintR5checker and conducted comprehensive experiments to check both the rule basis and program of Zlint. In the whole experiment we focus on rules expressing absolute requirements or prohibitions since flexible and truly optional rules are not required strictly.

4.1. Setup
Table 1 shows the hardware and software configurations, and Table 2 shows the Zlint commands which are invoked in the experiment.

| Item     | Configuration                          |
|----------|----------------------------------------|
| hardware | Intel Core i9-10900K, 32GB memory, and 500GB SSD |
| software | Ubuntu 20.04x64, Golang v1.16.4, Zlint v3.1.0, and Python v3.8.5 |

Table 2. Invoked commands and usages.

| Command | Usage                                      |
|---------|--------------------------------------------|
| zlint –includeSources RFC5280 –list-lints-json | obtain a set of lint related to RFC 5280 |
| zlint –includeSources RFC5280 –longSummary file.pem | obtain a summary of analytical results |
| zlint –includeSources RFC5280 –pretty file.pem | obtain a detailed analytical result |

In Table 2, the first command produces a JSON element of the lint set $I_{Zlint}$ in Line 1 of Algorithm 1. For example, \{"name":"e_issuer_field_empty","description":"Certificate issuer field MUST NOT be empty and must have a non-empty distinguished name","citation":"RFC 5280: 4.1.2.4","source": "RFC5280"\} is an output. Values of the key “description” constitute the rule set $R_{Zlint}$ in Line 3 of Algorithm 1. The second command outputs numbers of four levels (i.e., “info”, “warn”, “error”, and “fatal”) and corresponding lint names. The third command outputs analytical results i.e., $o_r$ in Line 3 of Algorithm 2. The format of $o_r$ is “lint name {‘result’: level}” e.g., “e_issuer_field_empty {‘result’: ‘error’}”. $R_r$ in Line 4 of Algorithm 2 is arrived at finding rules in the set of lint by lint names.

4.2. Rule discrepancies between Zlint and RFC 5280
According to Algorithm 1, 100 elements of $I_{Zlint}$ are extracted from Zlint. Based on the lint set $I_{Zlint}$, each sentence includes one key word which is specified in RFC 2119 is supposed to be a rule. Thus, discrepancies between Zlint and Section 4 of RFC 5280 are shown in Figures 2 and 3.

Figure 2. Classification of lint.

Figure 3. Intersection of RFC 5280 and Zlint.

Figure 2 shows that the lint set of Zlint is classified into 6 categories: (1) “Zlint-OK” denotes the lint has key words specified in RFC 2119 and all key words are capitalized, e.g., “basicConstraints MUST appear as a critical extension”; (2) “Zlint-mixedLU” denotes the lint has both uppercase and lowercase key words, e.g., “Email must not be surrounded with ‘<>’, and there MUST NOT be trailing comments in ‘()’”; (3) “Zlint-L” denotes the lint has only lowercase key words, e.g.,
“Certificate signature field must match TBSCertificate signature field”; (4) “Zlint-noKW” denotes the lint has no key words, e.g., “Internationalized DNSNames punycoded not valid unicode”; (5) “Zlint-nonR5” denotes the lint uses rules specified in other RFCs instead of RFC 5280, e.g., “RSA: Encoded public key algorithm identifier MUST have NULL parameters”; and (6) “Zlint-R5a” denotes the lint form rules with key words according to the appendix of RFC 5280, e.g., “The 'GivenName' field of the subject MUST be less than 17 characters”. The ratios of 6 categories in Figure 2 show that more than half the RFC5280-based lint does not strictly conform to RFCs 5280 and 2119.

Figure 3 shows rule relation of Section 4 in RFC 5280 to Zlint. Section 4 in RFC 5280 employs 152 rules to specify the syntax and semantics of certificates. Among these rules, 87 ones are adopted by Zlint. Note that 19 rules owned exclusively by Zlint are related to the categories “Zlint-nonR5” and “Zlint-R5a” shown in Figure 2.

The experimental results above show that the rule basis of Zlint could be more strictly standardized so as to output more strictly analytical results.

4.3. Issues existing in the program of Zlint

Based on the common rules shown in Figure 3, counter examples i.e., certificates violating these rules are generated to test the program of Zlint. According to Algorithm 2, rules related to the output of Zlint are compared with the rule employed to generate the certificate to detect issues with the usage of rules in the program of Zlint. Among the findings, two typical ones are as follows.

Finding 1: For a certificate without the extension “authority key identifier (AKI)”, Zlint reports two errors i.e., “e_ext_authority_key_identifier_missing” and “e_ext_authority_key_identifier_no_key_identifier” according to the rule “CAs must (MUST) support key identifiers and include them in all certificates” and the rule “CAs must (MUST) include keyIdentifier field of AKI in all non-self-issued certificates”. However, this certificate is generated by a rule similar to the former rule. It is not necessary to report the second error since it depends the existence of the extension AKI. This issue shows that the program of Zlint could use rules more intelligently.

Finding 2: For a version-1 certificate with the field “issuer unique identifier”, Zlint reports two contradictory errors i.e., “e_cert_contains_unique_identifier” and “e_cert_unique_identifier_version_not_2_or_3” according to the rule “CAs MUST NOT generate certificate with unique identifiers” and the rule “Unique identifiers MUST only appear if the X.509 version is 2 or 3”. The contradictory report comes from the contradictory rules.

4.4. Missing functions

Not all rules which are specified in RFC 5280 but not used by Zlint indicate missing functions. For example, the rule “conforming implementations MUST recognize version 3” specified in RFC 5280 cannot be checked by Zlint. However, some missing rules could be checked by Zlint. For example, the rule “It (serial number) MUST be unique for each certificate issued by a given CA” specified in RFC 5280 is not found in Zlint while the rule “Conforming CAs MUST NOT use serialNumber values longer than 20 octets” is found in Zlint. The missing rule indicates that Zlint cannot fulfil the function of checking the uniqueness of serial number. There are other missing functions. For example, a function corresponding to the rule “CAs conforming to this profile MUST always encode certificate validity dates through the year 2049 as UTCTime” is found in Zlint but a function related to the rule “certificate validity dates in 2050 or later MUST be encoded as GeneralizedTime” is missed. The function of checking GeneralizedTime-encoded years after 2050 is missing.

5. Related work

Guided by RFC 5280, RFCcert [6] generates certificates to test certificate validation. Zlint is a tool different from RFCcert since it checks certificates whether they violate rules of RFC 5280 and other specifications. Zlint itself provides some test cases in its source package. However, these test cases are not comprehensive enough since we found some issues and missing functions in invoking Zlint to do
research. Therefore, we put forward the approach ZlintR5checker for testing Zlint based on RFC 5280. ZlintR5checker checks both the rule basis and program of Zlint.

6. Conclusion
The correctness of X.509 certificate linter Zlint is very important since Zlint inspects the compliance of certificates which are critical to the security of Internet. Therefore, we propose ZlintR5checker, an approach for testing Zlint, from its rule basis to its program. With ZlintR5checker, rule discrepancies between Zlint and RFC 5280, issues with the usage of rules in the program of Zlint, and missing functions have been detected. Hence, our approach ZlintR5checker is effective in helping to improve the correctness of Zlint.

We plan to extend our approach from RFC 5280 to other specifications in the near future. Through future work, we look forward to improving the correctness of Zlint, enhancing the standardization of certificates, and strengthening the security of Internet.

Acknowledgments
This work is supported by Shandong Provincial Natural Science Foundation, China (ZR2020MF030, ZR2018PF007), CERNET Innovation Project (NGII20190407), and Natural Science Foundation of Xi’an University of Technology (413619001).

References
[1] Chen, Y, Su, Z. (2015) Guided differential testing of certificate validation in SSL/TLS implementations. In: the 2015 10th Joint Meeting on Foundations of Software Engineering. Bergamo, pp.793-804.
[2] Tian, C, Chen, C, Duan, Z, Zhao, L. (2019) Differential testing of certificate validation in SSL/TLS implementations: An RFC-guided approach. ACM Transactions on Software Engineering and Methodology, 28, 4: 24:1-24:37.
[3] Zlint. (2021). Zlint v3.1.0. http://github.com/zmap/zlint.
[4] Cooper, D, Santesson, S, Farrell, S, Boeyen, S, Housley, R, Polk, W. (2008) Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. http://rfc-editor.org/rfc/rfc5280.txt.
[5] Bradner, S. (1997) Key words for use in RFCs to indicate requirement levels. http://rfc-editor.org/rfc/rfc2119.txt.
[6] Chen, C, Tian, C, Duan, Z, Zhao, L. (2018) RFC-directed differential testing of certificate validation in SSL/TLS implementations. In: the 40th International Conference on Software Engineering. Gothenburg. pp. 859-870.