On the Complexity of the Web's PKI: Evaluating Certificate Validation of Mobile Browsers

Meng Luo, Bo Feng, Long Lu, Engin Kirda, and Kui Ren, *Fellow, IEEE*

***Abstract*—Digital certificates are frequently used to secure communications between users and web servers. Critical to the Web's PKI is the secure validation of digital certificates. Nonetheless, certificate validation itself is complex and error-prone. Moreover, it is also undermined by particular constraints of mobile browsers. However, these issues have long been overlooked. In this article, we undertook the first systematic and large-scale study of the certificate validation mechanism within popular mobile browsers to highlight the necessity of reassessing it among all released browsers. To this end, we first compile a comprehensive test suite to identify security flaws in certificate validation from various aspects. By designing and implementing a generic, automated testing pipeline, we effectively evaluate 30 popular browsers on two mobile OS versions and compare them with five representative desktop browsers. We found the latest mobile browsers `Accept` as many as 33.2% invalid certificates and `Reject` merely 5.4% invalid ones on average, leaving the majority of them to be decided by users who usually have little expertise. Our findings shed light on the severity and inconsistency of certificate validation flaws across mobile browsers, which are likely to expose users to MITM attacks, spoofing attacks, and so forth.**

***Index Terms*—The web's PKI, certificate validation, mobile browsers.**

I. Introduction

SECURE Socket Layer (SSL) and Transport Layer Security (TLS) protocols, coupled with the Public Key Infrastructure (PKI), are pervasively used within the Web [1], [2] to fulfill authenticity, confidentiality, and integrity of end-to-end communications. However, the actual use of certificates is intricate, and thus plenty of issues [3], [4], [5], [6], [7], [8], [9] have arisen and can lead users to man-in-the-middle (MITM) attacks, spoofing attacks, and so forth.

Securing the Web's PKI relies on not only trustworthy certificate issuance and management procedures but also reliable certificate validation mechanisms. Specifically, certificate validation is implemented by a web browser on behalf of the user to determine whether or not to trust a presented certificate for building a secure communication channel. The security community has previously paid attention to certificate security [8], [10], [11], as well as security issues with SSL/TLS implementations [12], [13], [14], [15], [16]. Nonetheless, certificate validation of browsers – especially mobile browsers – is strangely dismissed, even though they may contain many unique flaws. Moreover, there exist a surprisingly large number of certificates in the wild that are invalid [6] and distinguishing malicious certificates from other errors is extremely hard [17].

To fill this gap, in this work, we perform automated, large-scale testing on certificate validation of mobile browsers. To this end, we have to tackle the following challenges. First, certificate validation is an extremely complex and error-prone process, guided by multiple RFC standards (e.g., [18], [19], [20], [21]) and CA/Browser baseline requirements [22]. Due to vague documentation and internal conflicts, it is hard to determine the best practice of certificate validation and compose a comprehensive test suite for identifying security flaws.

The second challenge comes from automated testing. Certificate validation brings about additional latency and bandwidth burden for browsers while loading web pages. Due to constrained resources, mobile browsers tend to selectively validate certificates, resulting in significant discrepancies across different browsers and devices. As such, it is necessary to leverage dynamic analysis to evaluate mobile browsers. However, the research challenge is how one can create a generic framework capable of launching security testing of certificate validation on a wide range of mobile browsers and OSes.

In this paper, we design, implement, and evaluate CVHunter – a generic testing pipeline – that is capable of automatically generating, verifying, and evaluating test certificates over mobile browsers, as well as identifying security flaws of certificate validation. Orthogonal to prior work [12], [15], [23], [24], we leverage the relevant standards and adopt a rule-based approach for creating an ample set of test cases focusing on security-critical bugs instead of every variant of invalid certificates. To verify test certificates specifically, we develop a baseline certificate validation tool, which is later applied to be compared with mobile browsers for the detection of problematic certificates.

Manuscript received 1 January 2023; revised 23 February 2023; accepted 8 March 2023. Date of publication 13 March 2023; date of current version 12 January 2024. This work was supported by the National Science and Technology Innovation 2030 of China – Major program of "New generation of artificial intelligence" under Grant 2020AAA0107700, in part by the Key Program of the National Natural Science Foundation of China under Grant 62032021, in part by the Key Research and Development Program of Zhejiang Province, China under Grant 2019C03133, in part by the Key Laboratory of Zhejiang Province, China, and in part by the US NSF/CNS-2127200. *(Corresponding author: Bo Feng.)*

Meng Luo and Kui Ren are with the School of Cyber Science and Technology & ZJU-Hangzhou Global Scientific and Technological Innovation Center, Zhejiang University, Hangzhou, Zhejiang 310007, China (e-mail: meng.luo@zju.edu.cn; kuiren@zju.edu.cn).

Bo Feng is with the School of Cybersecurity and Privacy, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: bfeng64@gatech.edu).

Long Lu and Engin Kirda are with the Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115 USA (e-mail: l.lu@northeastern.edu; e.kirda@northeastern.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2023.3255869, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2023.3255869

We evaluate CVHunter on five major desktop browsers (on Windows) and thirty most popular mobile browsers, such as Chrome, Firefox, and Brave, using two operating systems (Android 6 and 10). We found that the latest mobile browsers could `Accept` as many as 33.2% invalid certificates and `Reject` merely about 5.4% invalid ones on average. The worst mobile browsers even blindly `Accept` nearly all invalid certificates. Next, the security of certificate validation is found to downgrade by changing from desktop to mobile or reverting the operating system. It is worth noting that mobile browsers are more conservative than desktop browsers to `Reject` invalid certificates meaning that `Warning` pages are favored for alerting potential risks. Finally, we investigate the behavior of browsers for handling invalid certificates in the wild.

The main contributions of this paper are summarized as follows.

- We compiled a comprehensive test suite – consisting of 157 test cases from five categories – by understanding the rules of multiple RFC standards and CA/B baseline requirements, as well as consulting prior work.
- We designed and implemented a certificate validation tool to verify test certificates and compared it to various popular mobile browsers.
- We designed and implemented an automated and generic testing pipeline for mobile browsers. *To the best of our knowledge, we are the first to systematically evaluate the certificate validation logic of mobile browsers.*
- Our findings shed light on the severity and inconsistency of security flaws in certificate validation across mobile browsers. Due to the widespread use of mobile browsers, we highlight that incorrectly handling invalid certificates can expose users to MITM attacks, spoofing attacks, and so forth. Consequently, a detailed and concrete guideline is needed to formalize a browser's behavior towards different types of invalid certificates.

The remainder of this paper is organized as follows. Section II introduces background knowledge. The test suite is presented in Section III. Section IV illustrates an automated evaluation pipeline. In Section V, we demonstrate our evaluation results of certificate validation on mobile browsers. Section VI analyzes problematic certificates in the wild. Finally, we discuss the related work and conclude this paper.

II. Background

In this section, we give a brief introduction to the Web's PKI, certification validation mechanism, as well as the relevant standards that guide certificate validation.

A. Web's PKI

Today, the Web's Public Key Infrastructure (shorted as the Web's PKI) plays a vital role in our daily lives. It secures a wide range of online activities (e.g., shopping, banking, *etc.*) by facilitating authentication and encrypted communication over insecure public networks using digital certificates and public-key encryption techniques.

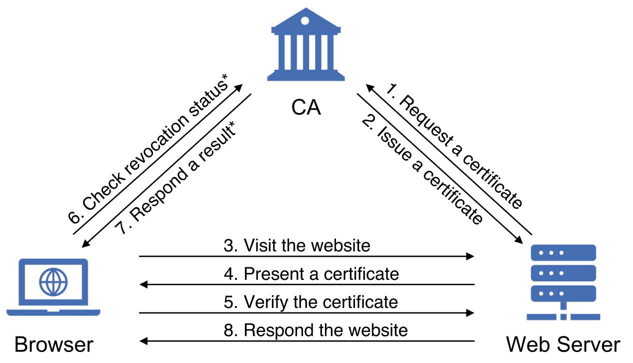

Fig. 1. A simplified illustration of the Web's PKI.

The security and privacy of the Web's PKI require not only secure issuance and management of certificates by Certificate Authorities (CAs) and web servers, but also reliable certificate validation mechanisms provided by browsers. As illustrated in Fig. 1, CAs take the responsibility of authenticating the identity of websites (or web servers), and then issue certificates. When necessary, web servers present the issued certificates for creating secure TLS connections with browsers. Before loading web pages through the TLS session, browsers need to verify the validity of presented certificates.

In case issued certificates are unwanted, CAs should revoke the certificates immediately to avoid possible security consequences. For example, the private keys of websites could be disclosed and abused to hijack communications. Also, CAs might be compromised by attackers to issue certificates maliciously as they want. In any case, information about revoked certificates should be propagated and notified to relevant entities, e.g., browsers. Currently, there exist multiple ways of querying certificate revocation status information, such as *Certificate Revocation List (CRL)* and *Online Certificate Status Protocol (OCSP)*. In addition, a few browser vendors have maintained some custom certificate revocation lists on their own. Therefore, querying certificate status may be done either offline or online during the certificate validation process.

B. Certificate Validation

Although attacks like man-in-the-middle may be prevented by using digital certificates, it is not trivial to establish a secure and reliable communication channel between the browser and web server. That is to say, it entails browsers to take rigorous certificate validation procedures while minimizing the page load time. Otherwise, it is by no means reasonable for browsers to trust certificates that could be potentially abused.

Certificate validation mainly involves: validating the chain of trust, which includes a sequence of certificates delegating the trust from one to another, checking the validity of all certificates, and matching the identity of the leaf certificate against a website's hostname. The chain of trust is built from a limited set of trusted root certificates, through intermediate CA certificates, to leaf certificates issued to websites. The root certificates are self-signed certificates that have already been added to the trust store of browsers or OSes. In [25], Zhang et al. investigated hidden root CAs imported to local stores but not in the public lists of the root CAs, which are usually maintained by mainstream OSes and browser vendors. However, detecting hidden root CAs is out of the scope of this paper.

The X.509 certificate provides various basic and extension fields for defining a certificate. To validate each certificate, browsers need to check *Signature* to verify the certificate's integrity, *Validity* to ensure the certificate does not expire, *Key Usage* to guarantee the usage of the certificate follows what it intends for, and the revocation status. Besides, certificate validation also involves verifying the chain of trust. Specifically, along with a certificate chain, there are multiple pairs of certificates where the former one is the *Issuer* of the latter one. For validating a certificate chain, browsers have to verify the trust relationship one by one for the key, name chaining, and enforcement of constraints. In name chaining, browsers verify that a certificate's *Subject* field is the same as the following certificate's *Issuer* field. In addition, to reduce risks, the certificate on a higher level of trust can impose constraints (e.g., path length constraints, name constraints, and policy constraints) over the subsequent certificates. Last, a leaf certificate's identity is matched against the identity of a website presenting that certificate to determine whether the certificate is used by the true owner. Note that sometimes more than one field is used to fulfill the same function, such as *Key Usage* and *Extended Key Usage*. As a result, certificate validation becomes a very complex and error-prone process.

C. Related Standards of Certificate Validation

RFC 5280 [18] and CA/B baseline requirements (BR) [22] are the two primary standards we investigated. RFC 5280 defines the format of X.509 certificates and the certificate validation process in detail. The CA/B BR is drafted by the CA and Browser forum, organized by major CAs and browser vendors, to guide the practical use of certificates. Apart from that, a few other standards help to supplement the certificate validation. RFC 6125 [19] defines the procedure of representing and verifying the identity of domain-based application services. RFC 6960 [20] defines the way of handling certificate revocations based on the *Online Certificate Status Protocol*. RFC 6962 [21] defines certificate transparency. Multiple reasons can hinder the correct validation of digital certificates. On the one hand, securely implementing the certificate validation mechanism requires browser developers to thoroughly investigate a wide range of standards and be aware of potential security hazards. It will be a huge amount of effort. On the other hand, as the X.509 certificate evolves, there might be some internal inconsistencies within the X.509 certificate. But, this problem has not attracted much attention. Finally, we refer to a wide range of relevant standards, online articles, and open discussions for getting a comprehensive understanding of the certificate validation process.

III. Test Suite Overview

To identify the noncompliance of certificate validation with related standards and detect potential flaws within different browsers, we compile a comprehensive and concise test suite by referring to multiple RFC standards [18], [19], [20], [21] (especially RFC 5280), CA/B baseline requirements [22], as well as existing studies [12], [15], [16], [24]. Specifically, we aim to make it not only comprehensive by covering a wide range of issues, but also concise by focusing on security-critical bugs. In this work, we try to generate certificates more likely to appear in the wild instead of making ad-hoc tests or involving simple policy violations (e.g., basic format requirements). In other words, we systematically investigated the semantic meaning and the requirements for certificate fields and then figured out potential violations, such as corner cases of values in (extension) fields and interplay among multiple relevant (extension) fields. CAs or browser developers are more likely to make such mistakes inadvertently. Therefore, the evaluation based on these test cases will have more security impact than the prior work, such as [12], [26]. As shown in Table I, our test suite consists of 157 test cases from five categories. We will introduce the philosophy of generating tests for each category as follows. More details regarding their potential threats are presented in the Appendix *(available online)*.

A. Key Intuition

We first investigate related resources listed above and then figure out a set of rules surrounding each test category. However, determining certificate validation rules is not trivial because of the obscurity of documentation, conflicts between different standards, and the introduction of new certificate fields sharing similar functions to existing ones. To tackle these challenges, we carefully search for and research the related online articles and discussions, besides standard documentation, to consolidate our expertise. Also, we define the principle of prioritizing security and balancing the ease of enforcement if necessary. To this end, we have to identify the strictest policies from multiple standards. We found conflicts commonly occur between RFC standards and CA/B BR and the policies of CA/B BR are usually stricter and more fine-grained. As CA/B BR is updated and agreed upon by major browser vendors and certificate authorities, we favor CA/B BR mostly when conflicts exist. To be cautious in determining rules, we also check the source code of reputable browsers, namely Chrome and Firefox.

Later, the rules we have determined are treated as seeds for generating tests. Using seed rules, we do not need to define the way of rule violations exhaustively for the specific tests but leverage them as core principles to produce security-related test cases instead. For example, the identity of certificates can be represented by two fields in X.509 certificate with different formats, and thus related requirements should be applied to handle diverse conditions. We develop a template-based system to derive particular test cases of rule violations and generate test certificates accordingly. To the best of our knowledge, we propose the first comprehensive test suite for evaluating certificate validation of browsers. More details on generating test certificates from the test suite are in Section IV.

B. Certificate Field Value

The X.509 certificate is structured by a few basic fields and extension fields (for *Version* 3 only). The fields all need to follow particular format and value requirements. Nevertheless, not all

TABLE I
An Overview of the Test Suite

Category	Content	#
Certificate field value	Aims to determine whether the requirements for the format and value of X.509 certificate fields are violated.	20
Certificate identity	Aims to evaluate whether the identity of certificates is represented under the correct field, and whether hostname verification runs properly on diverse conditions (*e.g.*, wildcard domain name).	26
Certificate chain validity	Aims to evaluate certificate chain validation which starts from root certificate to leaf certificate. Specifically, it covers diverse aspects such as name chaining, path length constraints, name constraints, period of validity, *etc*.	90
Key usage	Aims to identify whether the usage of certificates follows the scope and purpose (*e.g.*, server authentication, certificate signing) described in the certificate and whether there is inconsistency with definitions in the certificate.	13
Certificate revocation	As the use of maliciously or mistakenly issued certificates should be terminated, this category aims to detect whether revoked certificates are rejected correctly.	8

fields are considered security-related since we are not meant to capture every illegal value. For example, we exclude fields such as *Serial Number*. In addition, some fields are so important and complicated that we need to put them into separate categories. Finally, the fields checked here include:

1) *Version:* X.509 certificate version is either v1, v2, or v3. Extension fields are introduced in v3. RFC 5280 states that when extensions are used, the version must be v3. In contrast, the CA/B BR requires the version should be v3. As X.509 v3 certificate is prevalent and many security-related restrictions rely on extensions, we adopt the policy of CA/B. According to the rules, we produce tests by combining valid (i.e., v1, v3) or invalid (e.g., v5) versions, with or without extensions.
2) *Subject name:* Recognizing the correct identity of certificates is complicated and will be covered specifically. Here, we focus on the format of identity representation and the wildcard domain, which enables a certificate to be used by multiple domain names. RFC 6125 [19] specifies a few requirements for the use of the wildcard character in the domain name to prevent abuses. According to the rules, we craft diverse format violations regarding the identity representation and the wildcard domain to generate tests. For example, illegal positions of the wildcard character in the domain name (e.g., $*$.*co.uk*).
3) *Basic constraints:* The *basicConstraints* extension defines the role of certificates (via *cA*) and the maximum number of subordinate CA certificates on a certificate chain (via *pathLenConstraint*). According to RFC 5280 and CA/B BR, it is crucial for certificates to indicate their role between leaf and CA honestly. As for *pathLenConstraint*, leaf certificates must not include it, and meanwhile, CA certificates should follow particular formats for the value. According to the rules, we generate tests by varying the role of certificates and crafting diverse ways of role or path length value violating the true nature of the certificate.

C. Certificate Identity

Checking certificate identity is a critical step of certificate validation. Specifically, we need to deal with two things: figure out the correct identity of certificates, and verify whether the identity of a leaf certificate matches the website's identity.

The X.509 certificate provides two fields for certificate identity, namely *Subject* field and *Subject Alternative Name* extension. Especially, SAN enables a single certificate to associate with multiple domain names. RFC 5280 states that the identity may be carried in either *Subject* or *Subject Alternative Name* extension. Nonetheless, CA/B BR requires the *Subject* name must be contained in the *Subject Alternative Name* extension. In our opinion, there are many pitfalls around *Subject*, and thus we adopt CA/B BR's policy.

For *Subject*-only condition, we generate tests where the *Subject* field is different from the website's identity. Similarly, we generate tests for *Subject Alternative Name*-included condition and make *Subject* field be contained in *Subject Alternative Name*. In both conditions, certificate identity differs from the website's identity in that particular characters (e.g., wildcard, dot, *etc*.) are utilized improperly in different ways. Apart from that, we consider the condition where *Subject* field does not belong to the *Subject Alternative Name* extension. To generate tests, we choose to make either *Subject* or *Subject Alternative Name* not the same as the website's identity and then vary the value of the field similarly to previous conditions. The types of values in the certificate identity that are covered include fully-qualified domain name, wildcard domain name, IP address, and so forth. To distinguish from the previous category, here we use the wildcard character legally in the domain name.

D. Certificate Chain Validity

Certificate validation aims to verify the chain of trust starting from the trusted root certificates, via subordinate CA certificates, to a leaf certificate. Besides, the validity of every certificate along with the chain, such as the signature, period of validity, revocation status, and key usage, entails to be checked as well. To emphasize the importance of the key usage and certificate revocation, we prepare test cases for them especially, as shown in the rest of this section. Meanwhile, since certificate chain validation is a complicated process, we only present the most enticing parts and ignore some details (e.g., signature validation, long period of validity, and self-signed certificate) for simplicity.

1) *Name chaining:* It requires the *Subject* field of a certificate is the same as the *Issuer* field in the following certificate on the certificate chain. Any inconsistency occurred will break the trust of a chain. We generate tests by differentiating the above two fields and adding a few special characters (e.g., white space) that may be processed improperly by browsers, to *Subject* or *Issuer* fields.
2) *Path length constraints:* The *pathLenConstraint* is an attribute of *basicConstraints* extension and is used to define the maximum number of subordinate CA certificates following a CA certificate in the certificate chain. We generate tests that violate path length constraints of either

the root certificate or intermediate CA certificates. We design tests by varying certificate chain length, changing the position of rule violation, and manipulating the value of *pathLenConstraint*.

3) *Name Constraints:Name Constraints* extension is leveraged to place restrictions regarding subject namespace on any certificates sitting below that CA in the certificate chain, in the way of *Excluded Subtree* (a.k.a. blacklist), *Permitted Subtree* (a.k.a. whitelist) or both of them. The constraints may be enforced to *Subject* field or the *Subject Alternative Name* extension. The name forms, defined by RFC 5280, include *directoryName* for *Subject* field and *dNSName*, *iPAddress*, and so forth for *Subject Alternative Name* extension. From root certificate down to leaf certificate, if the subject name is not allowed according to *Name Constraints*, the certificate chain gets invalid.

We generate test cases for *Subject* and *Subject Alternative Name* respectively to evaluate the matching of the subject name against name constraints in conditions including whitelist-only, blacklist-only, as well as both of them are specified. Moreover, we vary our tests for different subject forms (e.g., IP address, fully-qualified domain name). For the domain name in particular, we complicate the checking by involving characters such as the wildcard, dot, etc. In addition, we include tests where the name constraints of either root CA or intermediate CA are violated, assuming that they are handled differently. For *Subject Alternative Name*, we also consider the case where multiple subject names are declared in *Subject Alternative Name*, and there exist conflicts in three ways. First, some of the subject names are not allowed by name constraints. Second, subject names are both permitted and excluded in the name constraints of one certificate (conflicts within a certificate). Third, subject names are both permitted and excluded in the name constraints of different CA certificates (conflicts between two certificates).

E. Key Usage

Depending on the role (e.g., CA or leaf), the public keys of certificates are used for diverse scopes and purposes. Two extensions, namely *Key Usage* and *Extended Key Usage*, are provided to regulate the usage of certificates. For example, a CA certificate can set *keyUsage* extension to *keyCertSign* to allow using the key to sign other certificates when a leaf certificate is not permitted to do so. Both *Key Usage* and *Extended Key Usage* extensions should follow some requirements made by CA/B BR and RFC 5280 to avoid making the capability of the certificate exaggerated. Specifically, particular values must be used in the key usage fields, but some others are forbidden, depending on the role of a certificate. Therefore, we first generate test cases by manipulating the values of the two extensions so that they violate those requirements. There are also extra restrictions for ensuring the consistency between *Key Usage* and *Extended Key Usage*. We then generate test cases to represent inconsistent purposes between *Key Usage* and *Extended Key Usage* in both leaf certificates and CA certificates, respectively.

F. Certificate Revocation

Attackers may abuse certificates maliciously to jeopardize users and website owners. For example, miscreants can compromise certificate authorities and then issue whatever certificates they want. Therefore, it is necessary to terminate the validity of these certificates before they expire. RFC 5280 and 6960 illustrate two approaches to query certificate revocation information, namely Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP). In addition, several factors may impact whether or not to check the revocation status, such as the position of the revoked certificate on the certificate chain and certificate chain length. Therefore, we generate tests for both CRL and OCSP by varying the certificate chain length, manipulating the role of the revoked certificate between CA and leaf, and changing the certificate status to be revoked.

IV. Automatic Testing of Certificate Validation

In this section, we present our methodology of testing certificate validation on a variety of popular mobile and desktop browsers. We first introduce how we obtain the dataset of browsers evaluated in this paper. Then, we demonstrate an automatic testing pipeline aiming to capture certificate validation flaws, regardless of the discrepancies across browsers.

A. Browser Dataset

To obtain an in-depth understanding regarding security issues of certificate validation, we select a list of the most popular mobile and desktop browsers, such as Chrome, Firefox, and Edge, as shown in Table II. This list is a representative one as it contains browsers using a variety of rendering engines and certificate validation mechanisms as well. Another consideration is that it is meaningful to compare mobile browsers with their desktop counterparts. Certificate validation means an additional non-negligible time overhead for browsers while loading web pages. Therefore, we would like to know whether and to what extent the security of certificate validation is sacrificed to improve a browser's performance.

Next, we discuss the main steps we took to collect the browser dataset. For mobile browsers, given the browser list obtained from Google Play Store, we try to collect the latest version of them. However, we need to roll back to earlier versions for a few browsers due to the following reason. According to prior work [27], the internal of mobile browsers can be determined by operating systems. Therefore, we are interested in observing the certificate validation mechanism of mobile browsers on different operating systems. To this end, we need to ensure that the latest browser versions are compatible with the evaluated operating systems.

Moreover, it is required by our automated testing framework to make tested browsers trust root certificates within the trust store of operating systems. As Firefox browser (100M+ installs) leverages a custom trust store, we decide to test Firefox beta instead, whose trust store is configurable. Finally, for desktop

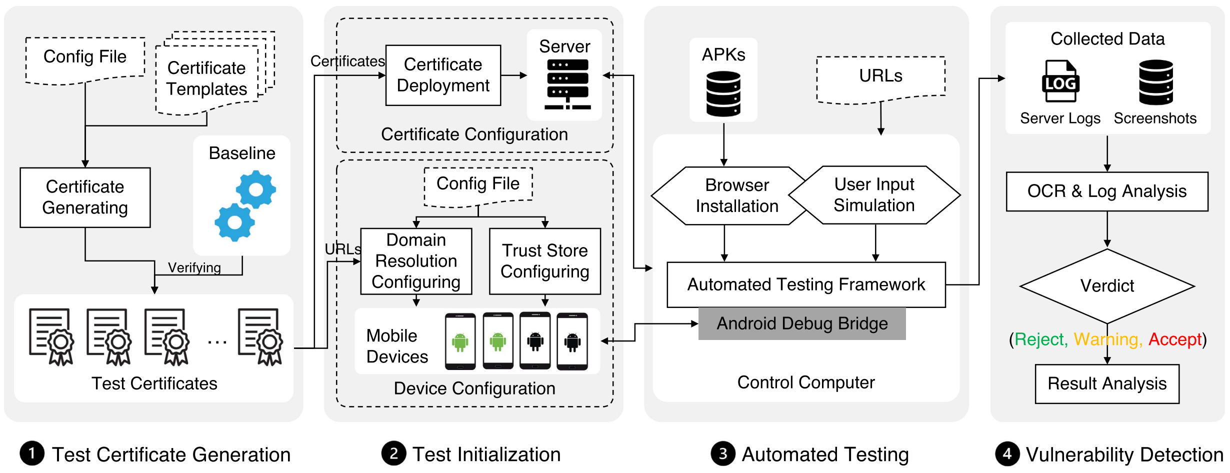

Fig. 2. Automatic testing pipeline for evaluating certificate validation of mobile browsers, which contains four steps: 1) *Test certificate generation:* produces and verifies test certificates, 2) *Test initialization:* configures testing environments, 3) *Automated testing:* performs testing, and 4) *Vulnerability detection:* identifies security flaws and analyzes results.

TABLE II
OVERVIEW OF SELECTED MOBILE AND DESKTOP BROWSERS

	Rank	Name	#Installs	Version
Mobile	1	Chrome	5,000M+	89.0.4389.86
	2	Samsung Internet	1,000M+	13.2.1.70
	3	UC Browser	500M+	13.3.8.1305
	4	Opera Mini	500M+	54.0.2254.56148
	5	Firefox Beta	10M+	87.0.0-beta.4
	6	Opera	100M+	62.3.3146.57763
	7	Phoenix	100M+	7.4.3.3060
	8	Mi Browser	100M+	12.10.5
	9	UC Mini	100M+	12.12.9.1226
	10	Browser	100M+	21.2.0.223
	11	Dolphin	50M+	12.2.5
	12	DuckDuckGo	10M+	5.77.2
	13	Edge	10M+	46.02.2.5147
	14	Brave	10M+	1.21.74
	15	UC Turbo	10M+	1.10.3.900
	16	Opera Touch	10M+	2.9.4
	17	MX5	10M+	5.2.3.3262
	18	Ecosia	10M+	4.4.0
	19	APUS Browser	10M+	3.1.4.1001
	20	Photon Browser	10M+	5.3
	21	Free Adblocker Browser	10M+	80.0.2016123375
	22	InBrowser	5M+	2.43-80
	23	Kiwi Browser	5M+	Git210216
	24	Via	5M+	4.2.6
	25	XBrowser	5M+	3.5.8
	26	Web Browser	5M+	3.5.7
	27	Web Explorer	5M+	4.3.3
	28	Lark Browser	5M+	1.1.26
	29	Search	5M+	4.4
	30	Lightning	1M+	5.1.0
Desktop	1	Chrome	-	89.0.4389.90
	2	Firefox Beta	-	87.0.0-beta.4
	3	Opera	-	76.0.4017.177
	4	Edge	-	91.0.864.41
	5	Brave	-	1.25.72

browsers, we choose the five most popular ones whose mobile counterparts are also in our browser list.

B. Automatic Testing Pipeline

To evaluate many non-cooperative browsers against hundreds of certificate-validation test cases, we have to cope with two challenges mainly. First, *how to transform test cases, as described in Section III, to some generic formats that could be easily tested by browsers?* Second, *how to enable us to perform automated and continued testing at scale for a variety of browsers, combined with diverse operating systems?*

To tackle these challenges, we design and implement an automatic testing pipeline aiming to identify certificate validation flaws in different browsers, which consists of *Test Certificate Generation*, *Test Initialization*, *Automated Testing*, and *Vulnerability Detection*. As testing desktop browsers is straightforward by leveraging browser automation tools like Selenium [28], we especially propose a generic testing framework for mobile browsers. We will illustrate the details of the automatic testing pipeline (shown in Fig. 2) using mobile browsers as the example.

Test Certificate Generation. It aims to transform test cases, described in Section III, to the outputs with a generic, browser-agnostic format, which are websites combined with test certificates. To this end, we propose a semi-automatic approach to generate test certificates and associated URLs from pre-defined templates. Our insight is that test cases deviate from valid certificates slightly by violating particular rules of standards. Therefore, the correctness of browsers for handling test certificates indicates whether browsers are secure or vulnerable. To fulfill the transformation, we first craft a few certificate templates. From there, we develop a certificate-generating system, which is used to produce various chains of test certificates by tweaking parameters and manipulating certificate templates. According to the requirements of test cases, the system generates chains of certificates, each of which is made by the root certificate, intermediate CA certificates, and a leaf certificate associated with a specific website. As a result, we output chains of certificates with corresponding URLs, which are later deployed for evaluating browsers.

Next, we develop a baseline certificate validation tool, which can perform certificate validation against test certificates similar to browsers. The baseline tool is based on the certificate validation process defined in RFC 5280 [18], and we add detailed security checks according to the test suite. We also refer to the source code of a few popular browsers like Chrome and Firefox. As expected, our baseline tool can reject all of the test certificates (explained further in Section V).

Test Initialization. It prepares the testing environment so that the testing framework can evaluate mobile browsers against test certificates as simple as visiting organic websites. The preparations include certificate configuration and testing platform configuration. The goal of certificate configuration is to deploy test certificates. For this purpose, we configure an Nginx web server to deploy test websites and certificates. As for testing platform configuration, we need to ensure that the testing platform (i.e., mobile devices) can satisfy all the requirements for the evaluation. More specifically, we first configure the *Dnsmasq* tool on mobile devices to forward requests towards domains associated with test certificates to the Nginx web server. Second, we configure mobile devices to install the root certificates that are the origin of trust for issuing all test certificates.

Automated Testing. It triggers automated testing framework to launch certificate validation testing on mobile browsers. It is cumbersome to manually evaluate thirty browsers on two mobile devices against a total of 157 test certificates. Hence, we design and implement an automated testing framework that is capable of automatically driving all tested browsers to visit test websites and then collecting necessary data. In this work, we evaluate mobile browsers on two Android OS versions, namely Android 10 and 6. The reason is that Android 10 is the most popular OS version at the time of testing, and we chose the next popular OS version–Android 6, to keep a gap between the two tested OS versions.

The automated testing framework controls mobile devices through Android Debug Bridge (ADB). Specifically, it first utilizes ADB commands to automatically install browsers (i.e., APK files) on mobile devices and mimic a sequence of user inputs (e.g., tap, swipe) to get through welcome pages. Then, the installed browsers are ready to be fed with URLs and perform testing. During the testing, the framework automatically provides simulated user interactions (e.g., tap, swipe) to rendered web pages and captures the screenshot of presented pages, as well as recording logs from the web server. Finally, the testing framework will instruct mobile devices to uninstall the tested browser and repeat the above steps for the next browser. Note that we spent a lot of effort to make the automated testing framework generic.

Vulnerability Detection. It analyzes collected web server logs and device screenshots, then automatically decides verdict results, which reflect decisions made by browsers. The verdict results are from either one of the following:

1) *Accept:* means a certificate is considered valid and trusted for establishing the communication channel.
2) *Warning:* means a certificate is invalid, but the problem is moderate. It lets users decide whether to proceed by displaying a warning page.
3) *Reject:* means a certificate is not trusted anyway, thus website visiting is stopped.

It is easy to know whether certificates are accepted or not, either by analyzing web server logs or presented web page content. However, it is hard to distinguish reject from the warning, in that they both show some browser-owned pages to stop users from directly accessing websites. In addition, extracting meaningful clues from presented pages is quite challenging even for human beings, not to mention that a general solution is needed for all browsers and diverse types of reject and warning reasons.

To address the problem, we try to collect browser-controlled error pages triggered by a wide range of reasons and provided by diverse browsers, leveraging our test websites. By extracting text clues from those pages using optical character recognition and comparing the differences, we generate an ample set of keywords for diverse types of error pages. Luckily, this process only needs to be done once, and the manual efforts are minimal. Given the collected data and pre-defined keywords, we apply both optical character recognition analysis and server log analysis to determine verdict results. Finally, we take further analysis over results to obtain more findings.

V. Analysis of Certificate Validation Problems

In our work, we evaluated 30 mobile browsers, five desktop browsers, and a baseline tool against 157 test certificates. Each mobile browser is evaluated on two different operating systems (Android 6 and 10). As a result, we performed a total of 10,362 tests automatically, each of which is labeled as either `Accept`, `Warning`, or `Reject`, indicating the decision made towards a test certificate. In the following, we will report our evaluation results in detail.

A. General Results

We compare evaluation results of certificate validation between baseline, mobile browsers, and desktop browsers in Fig. 3. Specifically, we count the total number of tests resulting in `Accept`, `Warning`, and `Reject` respectively. Unlike the baseline, there is not any browser that can `Reject` all test certificates. Moreover, `Reject` test certificates is not prevalent, most test certificates lead to `Warning` pages instead. Our results demonstrate that certificate validation flaws are indeed severe and inconsistent across different browsers.

We tested the desktop counterparts of five high-profile mobile browsers and compared the mobile and desktop side by side. It is shown that the security of certificate validation is impaired on mobile platforms. For example, average mobile browsers `Accept` more test certificates than desktop ones (about 20.9% for desktop browsers, and about 33.2% and 27.2% for Android 6 and 10 individually). In addition, average desktop browsers `Reject` significantly more test certificates than mobile browsers (37.6% for desktop versus 5.4% and 9% for Android 6 and 10). However, mobile browsers frequently display `Warning` pages and allow users to decide whether to trust a certificate. It is worth noting that `Warning` is not enough to prevent users from MITM attacks. According to previous work [29], up to 64.6% certificate warnings of Chrome on Android are clicked through, thereby potentially exposing users to MITM attacks. Hence, browser vendors should reconsider the decision of favoring `Warning` over `Reject` carefully.

Next, we found certificate validation of mobile browsers becomes slightly more secure due to system updates when we compared the same version of all browsers on two different operating systems. In other words, an average mobile browser elevates the decision of at least 13 test certificates to be `Warning` or `Reject` on Android 10 than Android 6. Last, certificate validation implementations vary a lot across mobile browsers,

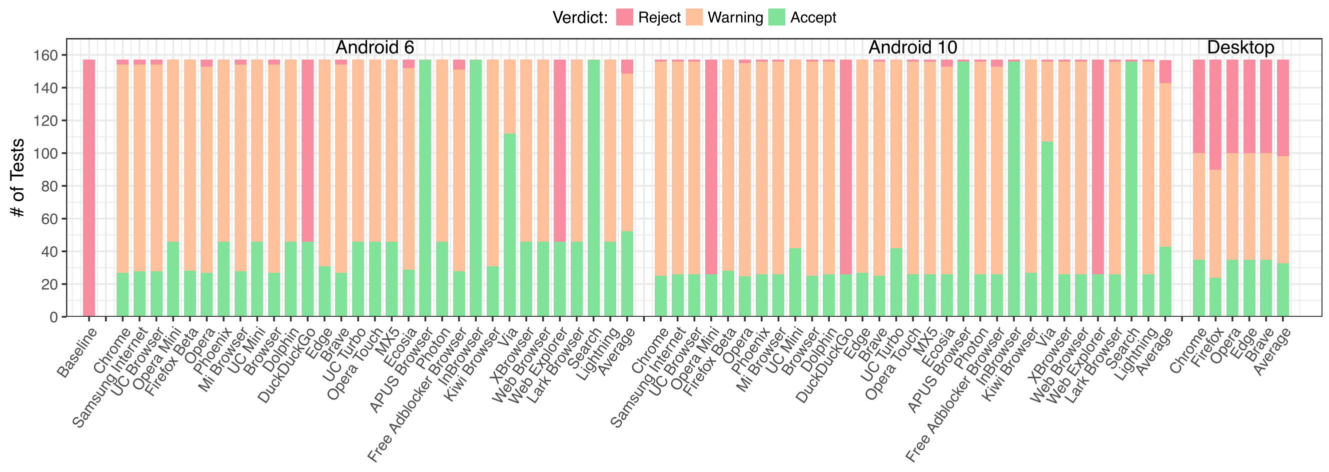

Fig. 3. Comparison of evaluation results for a baseline and diverse browsers (sorted by popularity) by varying underlying platforms and OSes.

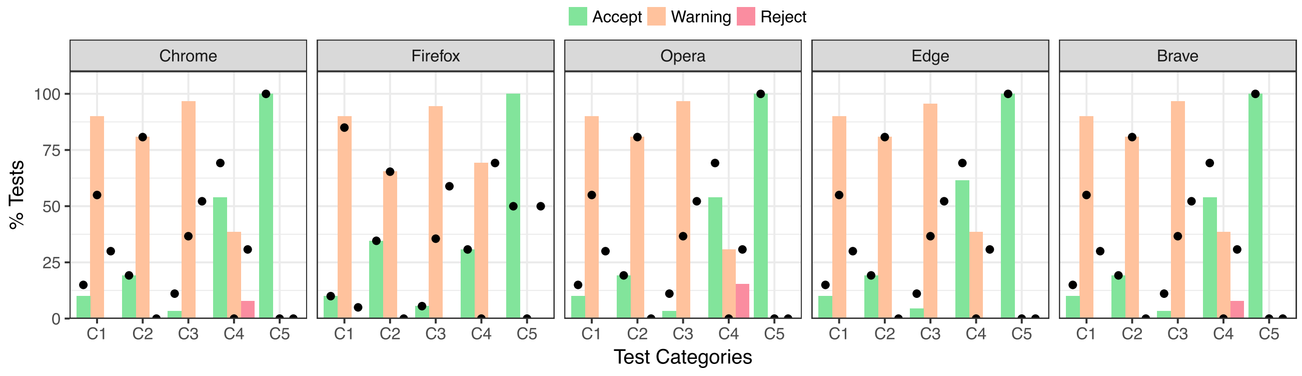

Fig. 4. Comparison of results between mobile (bars) and desktop (points) browsers. C1, C2, C3, C4, and C5 stand for different test categories, namely *certificate field value*, *certificate identity*, *certificate chain validity*, *key usage*, and *certificate revocation*.

whereas desktop browsers (except Firefox) show identical results. On Android 10, relatively more secure browsers (e.g., Opera Mini) `Reject` about 131 test certificates, whereas the most vulnerable browsers (e.g., APUS Browser) `Accept` nearly all test certificates.

B. Mobile versus Desktop Browsers

We found average desktop browsers `Accept` fewer invalid certificates than mobile browsers, indicating that desktop browsers care more about the security of certificate validation. We assume the reasons could be that either desktop devices are more powerful or the developers of mobile browsers tend to neglect security. Certificate validation entails extra time and bandwidth overhead for browsers while loading web pages. Desktop computers have enough resources (such as stable networks, abundant computing power, and so forth) for their applications, including web browsers. Therefore, desktop browsers can presumably leverage the ample resources to embed more secure and complete certificate validation algorithms than mobile ones.

In Fig. 4, we compare certificate validation results between five high-profile mobile browsers (on Android 10) and their desktop counterparts. We show the percentage of test certificates detected as `Accept`, `Warning`, and `Reject` in each test category, as described in Section III. To highlight the difference, we use bars and points to represent mobile and desktop browsers, respectively. Later, we define security levels – from low to high – as vulnerable (`Accept`), moderate (`Warning`), and secure (`Reject`). Security improvement indicates the security level changes from low to high, whereas security regression changes from high to low.

Counter-intuitively, Firefox is the only browser that improves (or at least not downgrades) security on the desktop for all categories. However, security improvements are still significant, and security regressions are minimal for other browsers. Firefox on desktop decides to `Reject` a significant amount of test certificates that used to be `Accept` or `Warning` on mobile browsers. Especially, Firefox for desktop is the only browser that correctly supports the checking of revocation status for leaf certificates (via OCSP) and `Reject` revoked certificates. That demonstrates how certificate validation gets improved on

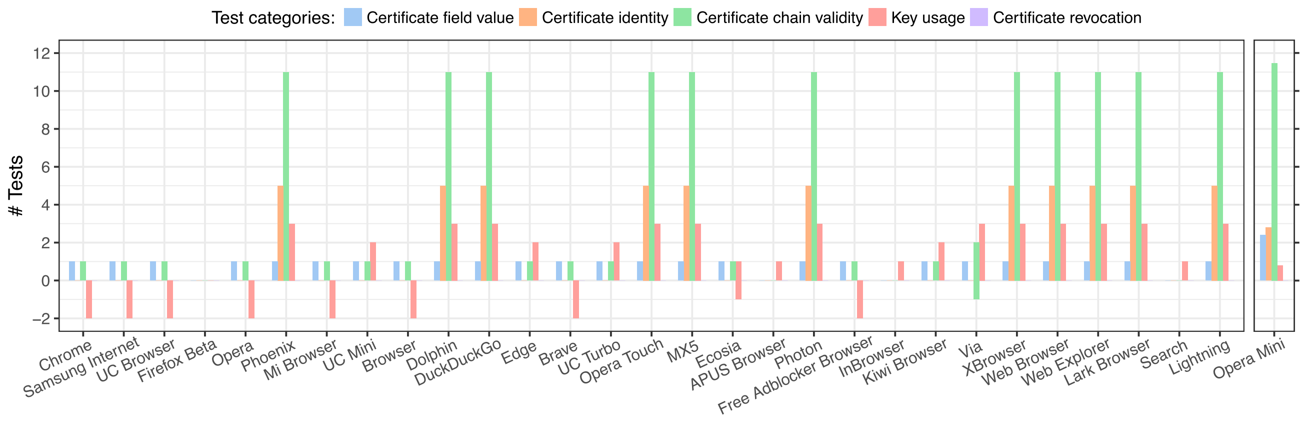

Fig. 5. Total number of tests with security improvement (positive on y-axis) or regression (negative on y-axis) for mobile browsers on Android 10 than Android 6; note that Opera Mini is placed separately due to scale difference.

desktop browsers prominently. Except for Firefox, other browsers have similar change patterns in that they make both security improvements and regressions. Common security downgrades include: using extensions but not configuring a correct certificate version (C1); violating name chaining or maximum period of validity (C3); failing to detect the inconsistencies between *keyUsage* and *extendedKeyUsage* (C4).

As for security improvements, desktop browsers tend to `Reject` more test certificates other than `Warning`, indicating that they are more confident about certificate validation results and not concerned about breaking websites. For example, desktop browsers (except Firefox) improve by `Reject` certificates declaring incorrect basic constraints, not satisfying path length constraints or name constraints, mistakenly allowing leaf certificates to sign certificates, and so on. Although desktop browsers make drastic improvements, a large number of flaws are still not eliminated fundamentally by both mobile and desktop browsers.

C. The Impact of Mobile Systems

Mobile browsers can integrate different "cores" depending on the operating system. Hence, even the same browser may exhibit differences in certificate validation from one OS to another. It is truly a problem for users who have difficulty updating the OS to the latest one on the market (e.g., due to hardware limitations). The reason is that these people presumably encounter certificate validation mechanisms with more vulnerabilities. As a result, it is necessary to understand how certificate validation is related to the OSes or mobile devices.

To this end, we tested mobile browsers on two mobile OSes (namely Android 6 and 10), which represent popular older and newer OSes. The security levels are defined the same as in the previous section. Security improvement means the security level changes from low to high, whereas security regression changes from high to low. As shown in Fig. 5, we demonstrate the total of tests with security improvement and security regression (for Android 10 over 6). We found all browsers, except Firefox, show some changes. The change patterns are attributed mainly to four groups and reveal the similarities within the group. The first one contains eleven browsers (e.g., Phoenix), the second is eight browsers (e.g., Chrome), the third is four browsers (e.g., UC Mini), and the last one is three browsers (e.g., APUS).

Ten browsers exhibit security regressions, but the number of cases is negligible (at most two tests per browser). By investigating further, we found one popular security regression is due to a mistake in handling an incorrect value of *Key Usage*; another one is because of the conflict between *Key Usage* and *Extended Key Usage*. It discloses that it is still confusing for browser vendors to determine certificate validation policies for dealing with the usage of keys.

Mobile browsers generally become more secure in certificate validation on newer OSes. Opera Mini achieves the most significant security improvements (131 test cases) among all browsers. Average mobile browsers make about 13 security improvements if all browsers are considered and nine when Opera Mini is excluded. Among five test categories, mobile browsers altogether make 222 (56.2%) security improvements in *Certificate chain validity*, 76 (19.2%) in *Certificate identity*, 54 (13.7%) in *Key usage*, 43 (10.9%) in *Certificate field value*, and 0 in *Certificate revocation*. For the specific security improvements, we obtain a few observations. First, *Certificate chain validity*, which involves a sequence of certificates, occupies more than half of all security improvements, meaning that the validation of certificate chains relies heavily on the computing power of underlying platforms. Second, mobile browsers achieve security improvements in approximately 13.9% of the tests in *Key usage* category. Therefore, it is valuable to interview browser developers to characterize the root causes of many uncertainties in regulating key usage. Last, certificate revocation checking is supported poorly by even the most high-profile browsers, and it probably results from the extra time and bandwidth cost.

D. Details of Non-Compliance With Standards

In this section, we discuss in detail the non-compliance of popular mobile browsers with standards (e.g., RFC 5280, CA/B BR). Since we tested the latest version of the most popular mobile browsers on the recent mobile OS, the non-compliances

TABLE III
GROUPING OF TEST RESULTS FOR BROWSERS ON ANDROID 10

	Browsers	Verdict	Total
G1	Opera	Accept	25
		Warning	130
		Reject	2
G2	Chrome, Browser, Brave	Accept	25
		Warning	131
		Reject	1
G3	Opera Mini, DuckDuckGo, Web Explorer	Accept	26
		Warning	0
		Reject	131
G4	Ecosia, Free Adblocker Browser	Accept	26
		Warning	127
		Reject	4
G5	Samsung Internet, UC Browser, Phoenix, Mi Browser, Dolphin, Opera Touch, MX5, Photon Browser, XBrowser, Web Browser, Lark Browser, Lightning	Accept	26
		Warning	130
		Reject	1
G6	Edge, Kiwi Browser	Accept	27
		Warning	130
		Reject	0
G7	Firefox	Accept	28
		Warning	129
		Reject	0
G8	UC Mini, UC Turbo	Accept	42
		Warning	115
		Reject	0
G9	Via	Accept	107
		Warning	49
		Reject	1
G10	APUS Browser, InBrowser, Search	Accept	156
		Warning	0
		Reject	1

of certificate validation appearing on Android 10 represent the most common flaws within mobile browsers nowadays.

In Table III, we demonstrate different groups of mobile browsers that exhibit identical results. We discovered that mobile browsers `Accept` more than 25 (16%) test certificates, even for relatively secure groups (G1-G3). Some mobile browsers (G10) even blindly `Accept` nearly all test certificates. Moreover, it is rare for mobile browsers (only G3) to `Reject` test certificates, although `Warning` is not the best choice. In the following, we will discuss non-compliances in each test category individually, according to Table IV.

Certificate Field Value. We carefully choose certificate fields that may lead to security issues once format or value violations exist. First, we found G7 (Firefox) `Accept` test certificates containing extensions, regardless of a certificate's version (even though it is not a valid one). Second, all browsers (except G7) do not check the correctness of both two attributes of *basicConstraints* in leaf certificates. Apart from that, G9 and G10 also fail to check *Subject Alternative Name* extension against the restrictions for using the wildcard character.

Certificate Identity. We identify rule violations of hostname verification, which checks the consistency between the identity of a certificate and the website that presents that certificate. Especially, browsers have to determine the correct certificate identity from multiple candidate fields. Again, G9 and G10 can not detect problems in all test cases. However, most groups (G1-G6) only make two types of mistakes, though these flaws occur in all tested browsers. The first one is allowing private IP addresses in either *Subject Common Name* or *Subject Alternative Name* to serve as the identity. Another flaw is that certificates whose *SCN* is not contained in *SAN* result in being `Accept`-ed, which violates the CA/B BR. Finally, G7-G10 incorrectly `Accept` certificates with inconsistent identities in the hostname and *SCN* field, such as encountering matching failures while involving the public IP address, wildcard character, and so on).

Certificate Chain Validity. We identify flaws that may cause a broken chain of trust to be permitted, including violations of validity on a specific certificate and incapable of enforcing constraints exerted by a certificate over the next ones on the chain. Since issuing certificates with a long period of validity will increase the risk of being compromised, CA/B BR mandates a policy of cutting the maximum period of validity according to when a certificate is issued. Nonetheless, we found that G7, G9, and G10 do not enforce the rules at all. Instead, G3-G6 and G8 partially enforce the rules but wrongly `Accept` certificates with a long period of validity and are issued recently (after September 2020).

In addition, all browsers (except G7) do not respect particular *pathLenConstraint* and *NameConstraints* exerted by root certificates, thereby loosing the control over intemediate certificates. Also, they wrongly `Accept` certificates violating specific *NameConstraints* for *Subject Alternative Name* while G7-G10 wrongly `Accept` certificates violating specific *NameConstraints* for *Subject Common Name*. G8 frequently violates the rules for checking *NameConstraints* against *SCN*. For example, the *PermittedSubtree* of *NameConstraints* includes "com" (supposed to allow test.com), however certificates whose *SCN* is "$*$.test.com" get `Accept`-ed mistakenly. *Key Usage.* We identify flaws where browsers wrongly `Accept` certificates even though the actual purposes of keys may not follow the values in *keyUsage* or *extendedKeyUsage* extensions. First, all browsers `Accept` CA certificates and leaf certificates that violate the rules for *keyUsage*. For example, leaf certificates are allowed to support *keyCertSign* enabling them to issue certificates, which violates the standard. Similarly, all browsers (except G7) `Accept` CA and leaf certificates that violate the rules for *extendedKeyUsage*. For example, CA and leaf certificates are allowed to specify any value in *extendedKeyUsage*. Finally, all browsers do not `Reject` certificates presenting particular inconsistencies between *keyUsage* and *extendedKeyUsage*.

Certificate Revocation. We identify flaws where revoked certificates are still considered valid due to not checking certificate status. Surprisingly, we found all mobile browsers do not query certificate status information via OCSP or CRL links specified in the certificate. Considering some browsers may leverage custom CRL, we undertake an extra experiment where we choose 35 revoked certificates randomly from *crt.sh* (a platform showing the details of certificates) and evaluate how browsers deal with them. It turns out that only Firefox can detect 14 revoked certificates. This result aligns with prior work [30], which discovered Chrome's CRLSet can only cover 0.35% of all revocations in their dataset.

E. Discussions

Analysis of Findings. Invalid certificates, once abused by adversaries but not detected, may lead users to various types of attacks (e.g., MITM) and punch holes over secure communication channels. In general, we found all popular mobile browsers, each of which has been installed millions to billions

TABLE IV
DETAILED EXPLANATIONS OF COMMON NON-COMPLIANCES WITH STANDARDS

Category	Description of non-compliance cases	Group
Certificate field value	Accepting certificates with extensions regardless of certificate version	G7
	Not checking the value of *SAN* against the requirements for the wildcard character	G9-G10
	Accepting leaf certificates with incorrect *basicConstraints*	All except G7
Certificate identity	Allowing private IP address to serve as certificate identity	All
	Not respecting *SAN* when *SCN* is not contained in *SAN*	All
	Incorrectly matching the identities of *SCN* and the website in particular ways	G7-G10
Certificate chain validity	Accepting certificates with long period of validity exceeding maximum validity period	G3-G10
	Not respecting *pathLenConstraint* and *NameConstraints* exerted by root certificates	All except G7
	Accepting certificates violating particular *NameConstraints* for *SCN*	G7-G10
	Accepting certificates violating particular *NameConstraints* for *SAN*	All except G7
Key usage	Accepting CA and leaf certificates violating *keyUsage* in particular ways	All
	Accepting CA and leaf certificates violating *extendedKeyUsage* in particular ways	All except G7
	Accepting certificates with inconsistent purposes in *keyUsage* and *extendedKeyUsage*	All
Certificate revocation	Not checking certificate revocation by CRL or OCSP	All

of times, contain some vulnerabilities and fail to identify invalid certificates. In addition, browsers show significant disparities across different platforms (i.e., desktop versus mobile), different Android OS versions, and diverse types of browsers. Due to the differences, a browser may embed with a different browser engine, thereby bringing about discrepancies in the root certificate store, trust paths, and certificate validation mechanism. In addition, we discovered that mobile browsers tend to hand over the responsibility of handling invalid certificates to users by showing "warning" pages. We suspect it is because the network condition of mobile devices is usually poor and can hinder the procedures like certificate revocation checks.

The identified flaws in the certificate validation logic of popular browsers have significant security impacts since we focus on security-related certificate-validation policy violations, which are more likely to appear in the real world. Instead, two lines of prior work also tried to generate test cases for evaluating certificate validation mechanisms. They are either limited to some ad-hoc tests [23], [24] or stop at very simple security-related policy violations (e.g., accepting X.509 version 1 certificate) [12], [26] by mutating fields or altering chain sizes. In this work, we dived into the specific values (e.g., corner cases) of (extension) fields in the X.509 certificate and explored complex situations, such as the interplay between multiple (extension) fields, besides the meaningful test cases presented in prior work. Interestingly, Kumar et al. reported in [8] that the error within most mis-issued certificates in 2017 is *Subject CN not from SAN*, which is also a prevalent vulnerability in our evaluation results. We shared a few other types of vulnerabilities with [8] as well. It indicates that our method is effective in identifying security-related certificate validation flaws, which are probably leveraged by real-world certificates. However, users can hardly perceive such attacks because they require no user interaction (e.g., clicking buttons), and hence the potential risks are catastrophic.

Ethical Disclosure. The invalid certificates in our test suite can help reveal the security flaws of browsers and point out how attackers can bypass the certificate validation mechanism. That is most of our invalid certificates, once crafted or obtained from certificate authorities by accident, can increase the risks of MITM attacks (like spoofing attacks and information theft). Nonetheless, they are not the vulnerabilities used by attackers directly to intercept end-to-end communications in the wild. Therefore, they can not be weaponized to automatically compromise web browsers or hack into end-to-end communications on a large scale.

Limitations & Suggestions. Although we tried to conduct a systematic and comprehensive analysis of certificate validation mechanisms, our work still contains some limitations. First, we tested popular mobile browsers on Android devices instead of other mobile platforms like iOS. Since we found the browser engine of many mobile browsers is correlated with their underlying OSes, it would be interesting to see which platform is generally more secure in terms of certificate validation. Next, the certificate validation for EV or OV certificates is stricter than standard certificates, realized through the source code of reputable browsers and relevant standards. However, we only generated standard certificates since it entails a strict reviewing process to obtain EV or OV certificates, meaning that we have to provide some documents to prove we physically own a company or organization. Therefore, we leave it to our future work. Finally, we made it our future work to systematically investigate browsers' implementations of certificate transparency-related policies.

To enhance the security of certificate validation, we have the following suggestions: 1) Due to the complexity and difficulty of certificate validation, we will suggest browser developers separate high-level policies from the implementation of certificate validation so that identifying flaws in the certificate validation can become easier than before; 2) Since browsers rely on the network to fetch data for completing certificate validation, we would recommend browser vendors to optimize network requests or preload some data.

VI. ANALYSIS OF REAL-WORLD CERTIFICATES

Although we tried to compile a comprehensive test suite, we still wonder whether test certificates can reflect problematic certificates in the wild. Moreover, we would like to know how popular mobile browsers react to these certificates. To this end, we evaluated certificates in the wild, collected from the Alexa top websites by randomly choosing 10K from the top, middle, and bottom of the list, respectively. In this section, we discuss our main findings.

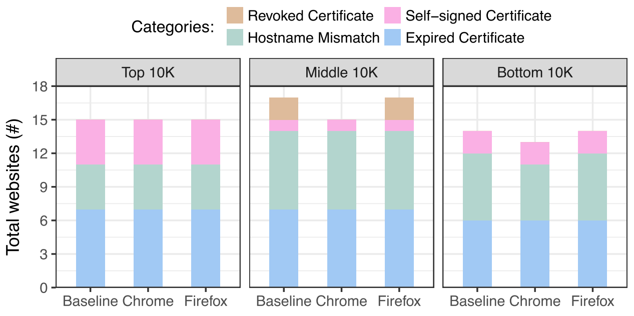

Fig. 6. Distribution of problematic certificates.

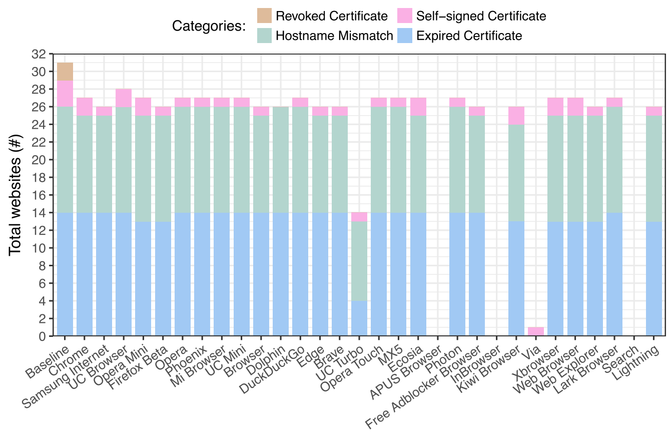

Fig. 7. Comparison between baseline and diverse mobile browsers for problematic certificate detection.

A. Problematic Certificates

From sampled websites, we utilize our baseline tool, Chrome, and Firefox (on Windows) to capture problematic certificates. In total, we collected 46 problematic certificates and classified them into four categories as the following:

1) *Expired Certificate:* the certificate presents when it exceeds (before or after) the period of validity.
2) *Hostname Mismatch:* the hostname of a web page does not match with the identity of the certificate.
3) *Self-signed Certificate:* the certificate is not signed by a trusted authority but an unknown entity or the owner itself.
4) *Revoked Certificate:* the certificate has been revoked due to some security concerns.

All these problems, found to be included in our test suite, are extremely harmful and may cause MITM attacks, spoofing attacks, and so forth. For example, the private keys of the self-signed certificate and expired certificate are at a higher risk of being disclosed, thereby causing hijacked communication channels. It further enables stealthy eavesdropping or modifying of private user information (e.g., bank accounts) and security-sensitive operations.

Fig. 6 shows the distribution of certificate problems across different website rankings. We found our baseline, the same as Firefox, detects the most number of real-world problematic certificates. The primary deficiency of Chrome is due to not detecting revoked certificates. Moreover, the Chrome browser adopts different policies for revoked certificates on Windows and macOS. For instance, users can access *mexicolibre.mx* (confirmed to have a revoked certificate by *crt.sh*) on Windows normally but are blocked on macOS.

We also analyze the relationship between website ranking and the total of problematic certificates. There is not enough evidence to prove that problematic certificates are more common for lower-ranked websites. However, we obtain a few interesting observations. First, renewing expired certificates is similarly challenging for websites across all rankings. Second, it is less common for users to encounter hostname mismatch issues if websites are high-ranked. Apart from ranking, we investigated website content associated with problematic certificates and found websites with problems, such as self-signed certificates or hostname mismatch, are more vulnerable. It is because the websites provide user accounts that are sensitive and may cause severe consequences once compromised.

B. Behaviors of Mobile Browsers

In this section, we present the results of evaluating the behavior of mobile browsers towards problematic certificates. Specifically, we evaluated the top 30 mobile browsers against the above 46 websites associated with problematic certificates. We found some websites either fixed their problems or became inaccessible anymore. As a result, 31 problematic certificates are still workable in the best case.

Fig. 7 demonstrates the total of problematic websites detected by different mobile browsers. Our baseline is proved to outperform all mobile browsers in terms of both the types and the total number of problematic certificates detected. Unlike our baseline, none of the mobile browsers can detect revoked certificates successfully. In addition, similar to the results in Table III, mobile browsers (e.g., APUS, Via), which fail to detect the majority of test certificates, still have difficulty in identifying the real-world problematic certificate. However, the rest of the mobile browsers show almost identical results. Interestingly, UC Turbo and UC Mini (previously in the same group) show significantly different results. According to further investigation, we found the results of UC Mini are impacted by enabling or disabling the data saving mode. Only by enabling the data saving mode, UC Mini is able to use proxy servers to achieve vital security mechanisms of browsers (e.g., certificate validation).

VII. Related Work

A. Certificate Security

The problem of abusing digital certificates to launch man-in-the-middle attacks has gained much attention over the years [3], [4], [5], [6], [7], [8], [17], [31]. Multiple approaches have been presented to understand the deficiencies of certificate validation in SSL/TLS implementations. Brubaker et al. [12] proposed a methodology to identify flaws in SSL/TLS implementations by differential testing on synthetic certificates generated by using corpus of fields and values from the real certificate. In [26], Chen et al. generated certificates by mutating certificates through a chain mutator and a certificate mutator. Both papers concentrated on producing as diverse certificates as possible to improve coverage instead of revealing security-related issues. Chau et al. [15] applied symbolic execution to test certificate chain validation code of small SSL/TLS libraries. Delignat et al. [32] analyzed the compliance of CAs with the guidelines made by the CA/Browser Forum over time. The abstraction of the CA/B guideline document is coarse-grained where [32] extracted only the keywords for the requirements in the guideline, whereas we understood the semantic meaning of it. Sivakorn et al. [16] paid special attention to hostname verification and presented a testing framework based on automata learning algorithms. Assuming that flaws are absent in SSL/TLS libraries, He et al. [13] checked the incorrect use of SSL/TLS APIs. However, none of these projects generates test certificates by looking at the noncompliance with related standards, not to mention that the perspective from browsers is dismissed, which is the motivation of our work. There are also related works covering other aspects of certificate validation logic, such as TLS handshake protocols [14], [33], Android apps [34], [35], [36], User Interface [37], and TLS Interception proxies [38], [39], [40], [41]. In addition, as ensuring the security of SSL/TLS implementations is extremely difficult, [42], [43], [44], [45], [46] demonstrated different approaches for mitigating the threats.

In recent years, there raises a concern regarding certificate revocation. In 2015, Liu et al. [30] conducted a measurement study for the checking of certificate revocation on different combinations of browsers and operating systems. They found the revocation check is very poorly supported. Compared to our work, [30] mainly focused on desktop browsers but not a diverse set of popular mobile browsers as we did. Also, it did not evaluate other certificate validation problems than certificate revocation. Apart from the efforts [47], [48] to overcome the shortcomings of traditional revocation status checking mechanisms, certificate transparency has recently been promoted to be effective in detecting malicious certificates. Nevertheless, Stark et al. [49] found the support of certificate transparency in browsers has been delayed.

B. Certificate Validation of Browsers

In 2009, Wazan et al. [23] were the first to evaluate browsers' behavior while processing certificates and demonstrated they are inconsistent, probably due to the ambiguity of standards. In 2017, the authors performed another evaluation by updating the test suite and adding a few more browsers. Then, they presented security improvements and regressions in [24]. The first difference between our work and [23], [24] is that they only cover several manually crafted test cases. Instead, we compiled a more updated and comprehensive test suite, which covers broad categories of invalid certificates, based on a systematic investigation of relevant standards and other resources. Second, our work is the first to concentrate on mobile browsers and to perform automated testing on dozens of the most popular mobile browsers. It allows us to reveal many mobile-specific issues.

As certificate validation mechanism is complex, Larisch et al. [50] proposed an approach to disentangle X.509 certificate validation policy (high-level rules) from the mechanism (low-level implementation code that enforces policies) and developed a pluggable framework called Hammurabi to replace the certificate validation mechanism of existing browsers. In addition, by leveraging the imputation techniques, developers can also discover differences between two browsers' certificate validation policies, thereby identifying security bugs.

VIII. Conclusion

Certificate validation plays a crucial role in establishing secure HTTPS connections. Due to the limitations of mobile platforms, it is challenging for mobile browsers to implement complete and secure certificate validation mechanisms. However, little attention has been paid to this area. To fill the gap, we performed a systematic and large-scale study for the certificate validation of mobile browsers. We first compiled a comprehensive test suite covering five aspects of certificate validation. Then, we developed an automated testing pipeline and leveraged it to evaluate thirty mobile browsers and five desktop browsers against a total of 157 test cases.

The evaluation reveals that desktop browsers are more secure than mobile browsers since they prefer rejecting invalid certificates instead of showing warning pages. Interestingly, mobile browsers can achieve significant security improvements by updating OSes. Our findings are confirmed via the analysis of problematic certificates on real websites. To our best knowledge, we are the first to systematically evaluate certificate validation mechanisms of mobile browsers. We highlight mobile browsers are as much important, if not more, than desktop browsers and other SSL/TLS implementations.

References

[1] Google, "HTTPS Encryption on the Web," 2022. Accessed: Jan. 2022. [Online]. Available: https://transparencyreport.google.com/https/overview
[2] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS adoption on the web," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1323–1338.
[3] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The SSL landscape: A thorough analysis of the X. 509 PKI using active and passive measurements," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf.*, 2011, pp. 427–444.
[4] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer, "Here's my cert, so trust me, maybe? Understanding TLS errors on the web," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 59–70.
[5] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *Proc. Conf. Internet Meas. Conf.*, 2013, pp. 291–304.
[6] T. Chung et al., "Measuring and applying invalid SSL certificates: The silent majority," in *Proc. Internet Meas. Conf.*, 2016, pp. 527–541.
[7] M. E. Acer et al., "Where the wild warnings are: Root causes of chrome HTTPS certificate errors," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1407–1420.
[8] D. Kumar et al., "Tracking certificate misissuance in the wild," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 785–798.
[9] V. Drury and U. Meyer, "Certified phishing: Taking a look at public key certificates of phishing websites," in *Proc. 15th Symp. Usable Privacy Secur*, USENIX Assoc., Berkeley, CA, USA, 2019, pp. 211–223.
[10] Q. Scheitle et al., "The rise of certificate transparency and its implications on the internet ecosystem," in *Proc. Internet Meas. Conf.*, 2018, pp. 343–349.

[11] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, "Bamboozling certificate authorities with BGP," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 833–849.
[12] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 114–129.
[13] B. He et al., "Vetting SSL usage in applications with SSLINT," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 519–534.
[14] J. Somorovsky, "Systematic fuzzing and testing of TLS libraries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1492–1504.
[15] S. Y. Chau et al., "SymCerts: Practical symbolic execution for exposing noncompliance in X.509 certificate validation implementations," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 503–520.
[16] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana, "HVLearn: Automated black-box analysis of hostname verification in SSL/TLS implementations," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 521–538.
[17] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 83–97.
[18] D. Cooper et al., "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," https://datatracker.ietf.org/doc/html/rfc5280, RFC Editor, RFC 5280, May 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5280
[19] P. Saint-Andre and J. Hodges, "Representation and verification of domain-based application service identity within internet public key infrastructure using X.509 (PKIX) certificates in the context of transport layer security (TLS)," https://datatracker.ietf.org/doc/html/rfc6125, RFC Editor, RFC 6125, Mar. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6125
[20] S. Santesson et al., "X.509 Internet public key infrastructure online certificate status protocol-OCSP," https://datatracker.ietf.org/doc/html/rfc6960, RFC Editor, RFC 6960, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6960
[21] B. Laurie et al., "Certificate transparency," https://datatracker.ietf.org/doc/html/rfc6962, RFC Editor, RFC 6962, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6962
[22] C. Forum, "Baseline requirements for the issuance and management of publicly-trusted certificates version 1.7.6," 2022. Accessed: Jan. 2022. [Online]. Available: https://cabforum.org/baseline-requirements-documents/
[23] A. S. Wazan, R. Laborde, D. W. Chadwick, F. Barrere, and A. Benzekri, "Which web browsers process SSL certificates in a standardized way?," in *Proc. IFIP Int. Inf. Secur. Conf.*, Springer, 2009, pp. 432–442.
[24] A. S. Wazan, R. Laborde, D. W. Chadwick, F. Barrere, and A. Benzekri, "TLS connection validation by web browsers: Why do web browsers still not agree?," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf.*, 2017, pp. 665–674.
[25] Y. Zhang et al., "Rusted anchors: A national client-side view of hidden root CAs in the web PKI ecosystem," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 1373–1387.
[26] Y. Chen and Z. Su, "Guided differential testing of certificate validation in SSL/TLS implementations," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 793–804.
[27] M. Luo, P. Laperdrix, N. Honarmand, and N. Nikiforakis, "Time does not heal all wounds: A longitudinal analysis of security-mechanism support in mobile browsers," in *Proc. 26th Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15.
[28] Selenium, "Selenium," 2022. Accessed: Jan. 2022. [Online]. Available: https://www.selenium.dev/
[29] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness," in *Proc. 22nd USENIX Secur. Symp.*, Washington, D.C.: USENIX Assoc., 2013, pp. 257–272. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe
[30] Y. Liu et al., "An end-to-end measurement of certificate revocation in the web's PKI," in *Proc. Internet Meas. Conf.*, 2015, pp. 183–196.
[31] M. Ukrop, L. Kraus, V. Matyas, and H. A. M. Wahsheh, "Will you trust this TLS certificate? Perceptions of people working in it," in *Proc. 35th Annu. Comput. Secur. Appl. Conf.*, 2019, pp. 718–731.
[32] A. Delignat-Lavaud, M. Abadi, A. Birrell, I. Mironov, T. Wobber, and Y. Xie, "Web PKI: Closing the gap between guidelines and practices," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–15.
[33] A. Walz and A. Sikora, "Exploiting dissent: Towards fuzzing-based differential black-box testing of TLS implementations," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 2, pp. 278–291, Mar./Apr. 2020.
[34] L. Onwuzurike and E. De Cristofaro, "Danger is my middle name: Experimenting with SSL vulnerabilities in android apps," in *Proc. 8th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2015, pp. 1–6.
[35] C. M. Stone, T. Chothia, and F. D. Garcia, "Spinner: Semi-automatic detection of pinning without hostname verification," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, 2017, pp. 176–188.
[36] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android SSL (in) security," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 50–61.
[37] K. Wang et al., "Assessing certificate validation user interfaces of WPA supplicants," in *Proc. 28th Annu. Int. Conf. Mobile Comput. Netw.*, 2022, pp. 501–513.
[38] X. D. C. de Carnavalet and M. Mannan, "Killed by proxy: Analyzing client-end TLS interception software," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–17.
[39] L. Waked, M. Mannan, and A. Youssef, "To intercept or not to intercept: Analyzing TLS interception in network appliances," in *Proc. Asia Conf. Comput. Commun. Secur.*, 2018, pp. 399–412.
[40] A. S. Wazan et al., "On the validation of web X.509 certificates by TLS interception products," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 227–242, Jan./Feb. 2022.
[41] B. Kondracki, A. Aliyeva, M. Egele, J. Polakis, and N. Nikiforakis, "Meddling middlemen: Empirical analysis of the risks of data-saving mobile browsers," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 810–824.
[42] A. Bates et al., "Securing SSL certificate verification through dynamic linking," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 394–405.
[43] D. Kaloper-Mer šinjak, H. Mehnert, A. Madhavapeddy, and P. Sewell, "Not-quite-so-broken TLS: Lessons in re-engineering a security protocol specification and implementation," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 223–238.
[44] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno, "Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 235–254.
[45] M. O'Neill et al., "TrustBase: An architecture to repair and strengthen certificate-based authentication," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 609–624.
[46] Z. Ma et al., "What's in a name? Exploring CA certificate control," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 4383–4400.
[47] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A scalable system for pushing all TLS revocations to all browsers," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 539–556.
[48] L. Chuat, A. Abdou, R. Sasse, C. Sprenger, D. Basin, and A. Perrig, "SoK: Delegation and revocation, the missing links in the web's chain of trust," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2020, pp. 624–638.
[49] E. Stark et al., "Does certificate transparency break the web? Measuring adoption and error rate," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 211–226.
[50] J. Larisch et al., "Hammurabi: A framework for pluggable, logic-based X.509 certificate validation policies," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 1857–1870.

Meng Luo received the BEng degree in information security from Wuhan University, Wuhan, China, in 2015, and the PhD degree in computer science from Stony Brook University, Stony Brook, NY, USA, in 2020. She is a ZJU 100-Young professor with the School of Cyber Science and Technology & ZJU-Hangzhou Global Scientific and Technological Innovation Center, Zhejiang University, Hangzhou, China since 2022. She was a post-doctoral research associate with the Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA. Her research interests include mobile security, web security, and IoT security.

Bo Feng received the bachelor's degree in computer science from Wuhan University, China, in 2015, and the PhD degree in computer science from the Khoury College of Computer Sciences, Northeastern University, USA, in 2022. He is a postdoctoral fellow with the School of Cybersecurity and Privacy, College of Computing, Georgia Institute of Technology, USA. His research interests include system security, IoT security, and embedded devices.

Long Lu is an associate professor with the Khoury College of Computer Sciences, a core faculty member of the Cybersecurity and Privacy Institute, and the co-director with the SecLab (Systems Security Lab), Northeastern University. His research aims to secure low-level software in widely deployed or critical systems. He designs and builds novel program analysis and hardening techniques, hardware-backed primitives for security, and trusted/confidential computing environments. His recent work has focused on embedded and IoT/CPS systems. He has won an NSF CAREER Award, an Air Force Faculty Fellowship, two Google ASPIRE Awards, etc. His research is supported by the National Science Foundation, the Office of Naval Research, the Army Research Office, etc.

Engin Kirda is a professor with the Khoury College of Computer Sciences and the Department of Electrical and Computer Engineering, Northeastern University in Boston. Previously, he was a tenured faculty with Institute Eurecom (Graduate School and Research Center) in the French Riviera and before that, faculty with the Technical University of Vienna where he co-founded the Secure Systems Lab. His lab has now become international and is distributed more than nine institutions and geographical locations. His current research interests include systems, software and network security (with focus on Web security, binary analysis, malware detection). Before that, he was mainly interested in distributed systems, software engineering, and software architectures. He is also a part of the Shellphish Hacking Group. They regularly participate at the DefCon CTF.

Kui Ren (Fellow, IEEE) received the PhD degree in electrical and computer engineering from Worcester Polytechnic Institute. He is a professor with Zhejiang University, where he also directs the Institute of Cyber Science and Technology. His current research interests include data security, IoT security, and AI security. He received many recognitions including Guohua Distinguished Scholar Award of ZJU, IEEE CISTC Technical Recognition Award, SUNY Chancellor's Research Excellence Award, Sigma Xi Research Excellence Award, NSF CAREER Award, etc. He has published extensively in peer-reviewed journals and conferences and received the Test-of-time Paper Award from IEEE INFOCOM and many Best Paper Awards. His h-index is 87, with a total citation exceeding 41,000 according to Google Scholar. He is a fellow of the ACM. He currently serves as the chair of SIGSAC of ACM China Council, a member of ACM ASIACCS steering committee, and a member of S&T Committee of Ministry of Education of China.