Do Web Browsers Obey Best Practices When Validating Digital Certificates?

A Thesis Proposal Presented
by

Krati Kiyawat

to the faculty of
College of Computer and Information Science

in Partial Fulfillment of the Requirements
for the Degree of

Master of Science in Information Assurance

Northeastern University
Boston, Massachusetts

December 2014

Contents

Abstract

The SSL/TLS protocol forms the basis of secure communication on the Internet. One of the key components of this protocol are digital certificates that allow clients to ascertain the identity of the remote party. Validating these certificates is critical before establishing a secure client-server connection. If certificate validation is not performed thoroughly and correctly, clients become vulnerable to a variety of attacks that can compromise the authenticity, integrity, and confidentiality of their communications. However, validation checking is a complicated task that involves local and remote information, and thus may slow down user's connections.

Given that there is no standard that defines how certificates must be validated, browser developers may choose to implement a subset of validation checks rather than obeying security best-practices. In this study, we present an evaluation of the behavior of modern web browsers when presented with different invalid certificates on different platforms. In order to perform the experiments, we set up an extensive test-bed of web-servers and certificates.

Once the testbed was successfully set up and verified, we examined the behavior of Google Chrome and Mozilla Firefox when presented with 26 different kinds of erroneous certificates. During our study, we identified many flaws in the targeted browsers. We observed different behavior of Google Chrome in different platforms, while Firefox exhibited uniform behavior on all the platforms tested. With an exception to Google Chrome on Windows for EV certificates, none of the browser implementation checks for CRLs. We also observed that the mobile versions of the browser are less secure than their desktop versions. Surprisingly, our testing revealed that Internet Explorer performs exhaustive revocation checking, by implemented OCSP and CRL checks as well.

In conclusion, we highly recommend that browser designers implement validation checks similar to Internet Explorer. Also, users who wanted to protect themselves online should prefer desktop versions of the browsers rather than mobile versions for security-critical communications.

Acknowledgments

Foremost, I would like to express my deepest gratitude to my advisor Prof Christo Wilson. This work wouldn't have been possible if he hadn't given me a chance to work with him. Since I approached him looking for an advisor, he has been a great support and helped me in every step shaping this thesis. His guidance helped me in all the time of research and writing of this thesis. He has always taken time out from his busy schedule to guide me in the right direction whenever I needed help during the course of this research.

I would also like to express my sincere thanks to Prof. Themis Papageorge, who has been guiding me throughout my Masters degree as a program adviser. Since I came to Northeastern University, he has been the best resource for structuring my degree in my best interest and helped me broaden my skills. Timely suggestions and support by both of them has made me a better person and researcher today. I feel humbled to have received guidance from people of great stature and will cherish this forever.

I finally would like to thank my family and friends who always have believed, supported and encouraged me to pursue my interests and passion in life.

Chapter 1

Introduction

The SSL/TLS protocol is used to provide security over the insecure Internet [1, 19]. The security provided by SSL/TLS is twofold: *first*, it allows a client to ascertain the identity of a remote party through the use of digital *certificates* tied to a Public Key Infrastructure (PKI). Certificates are signed by the trusted Certificate Authorities (CAs) that issue them; this allows clients to validate certificates by tracing their lineage back to a trusted CA. *Second*, SSL/TLS provides confidentiality and integrity between communication parties via encryption.

Many security-critical transactions are performed over the Internet, including banking, e-commerce, and email. In these cases, validating the authenticity of certificates is crucial before establishing a secure client-server connection. If the validation is not performed correctly, this leaves clients open to a variety of attacks that compromise the integrity and confidentiality of their communications. While in theory, all modern web browsers should perform stringent validation checks on certificates before establishing a secure connection, it is not clear if this occurs in practice.

Now, perhaps more than ever, it has become extremely important that browsers perform strict validation checking on certificates. In April 2014, a critical vulnerability called Heartbleed was exposed in the OpenSSL cryptographic library [11]. This vulnerability potentially exposed the private encryption keys of roughly 17% of the web servers on the Internet [20, 29]. To make matters worse, Heartbleed was present on the Internet for almost 2 years without being detected. In the aftermath of Heartbleed's disclosure, observers noticed a surge in certificate revocations in April 2014 [17]. Similarly, in 2008 a bug in Debian's pseudorandom number generator enabled attackers to guess the private keys on vulnerable servers [28].

Once the certificate of the server is compromised, that entity and the clients it serves become vulnerable to various attacks. The identity of the server can be easily spoofed and impersonated by an attacker that is in possession of the private key and the certificate. Moreover, all the communications encrypted with the compromised keypair can be decoded and read by the attacker using a man-in-the-middle attack [21]. Thus, private key compromise makes the need for certificate validation imperative.

Validating a certificate includes many individual checks. Some of these checks can be performed locally on the client just based on information contained in the certificate, such as: matching the subject name to the server's hostname, making sure the certificate is not expired, verifying that the certificate is not self signed, and validating the chain of cryptographic signatures from the leaf down to the root certificate. However, revocation checks incur additional overhead because the necessary information is stored on remote servers. To check if a certificate has been revoked (*i.e.* forcefully expired by its owner), the browser must download a Certificate Revocation List (CRL) or query an Online Certificate Status Protocol (OCSP) resolver, depending on how the given certificate has been configured.

Given that there is no standard that defines how certificates must be validated, browser developers may choose to implement a subset of validation checks rather than obeying security best-practices. Revocation checking is particularly contentious because it adds additional delays to the establishment of HTTPS sessions. It is known that Chrome does not support CRLs; instead, it only checks a subset of

all revocations contained with a CRLset file that is distributed by Google [8]. Mozilla has also recently disabled CRL checks in Firefox and is moving towards a similar architecture to Chrome [7]. Others have already noticed that many browsers fail to properly check the revocation status of certificates [2, 10, 12, 24].

In this study, we present an evaluation of the behavior of modern web browsers when presented with different invalid certificates on different platforms. Although we examine many different types of erroneous certificate chains, we specifically focus on the browser behavior with respect to revoked certificates.

In order to perform experiments on various browsers, we built an extensive testbed of web servers and certificates. Configuring the test environments consists of various steps including: selection of browsers and the platforms, generation of certificates, and configuration of the HTTPS services to host the certificates. In addition to desktop browsers, we also evaluated mobile browsers. We discuss the design of our testbed and our experimental methodology in detail in § 3.

Once the testbed was successfully set up and verified, we examined the behavior of Google Chrome and Mozilla Firefox when presented with 26 different kinds of erroneous certificates. While performing this analysis (presented in § 4) we discovered many flaws in the validation process of the target browsers. While Google Chrome produced diverse results depending on the underlying Operating System (OS), Mozilla Firefox exhibited uniform results across all platforms. We observed that Chrome's validation approach is buggy on Android and Windows. We also discovered that the mobile version of the browsers are less secure than the desktop versions.

Surprisingly, our testing revealed that Internet Explorer performs exhaustive revocation checking by first attempting OCSP validation, then falling back to CRLs. Neither Chrome nor Firefox implement CRL checks across all platforms and types of certificates. This suggests that other browser designers should also implement Internet Explorer's approach towards revocation checking.

We conclude in § 5 by discussing the implications of our findings. This includes concrete recommendations for users who want to protect themselves online, as well as browser developers looking to enhance the security of their software and better protect their users.

Chapter 2

Background

In this section, we review background information related to SSL/TLS, X.509 certificates, and certificate validation. We begin by briefly discussing the basics of the SSL/TLS PKI. Next, we examine Extended Validation (EV) certificates, followed by a discussing of certificate revocation. Finally, we present an overview of the correct steps a browser should take when validating a certificate. In the remainder of this study, we examine whether browsers actually obey these recommendations.

2.1 SSL, Certificates, and PKI

Secure Socket Layer (SSL) / Transport Layer Security (TLS) is the transport layer security protocol designed to provide authenticity, integrity, and confidentiality over the public Internet. The SSL/TLS protocol is the precondition for secure online transactions in services like e-commerce, banking, and email. The older SSL protocol has been found to be vulnerable to numerous attacks [26, 27], and thus it is currently being phased out in favor of the more advanced and secure TLS protocol.

One of the fundamental components of the SSL/TLS protocol is digital certificates. Digital certificates are associated with two encryption keys: a public key available to everyone that is embedded inside the certificate, and a private key known only to the certificate owner. The public key establishes the identity of an entity. The public-private key pair is used to implement asymmetric encryption between parties using SSL/TLS, and thus insure the confidentiality and integrity of communications. SSL/TLS uses a client–server paradigm, where the server authenticates itself by presenting its digital certificate to the client. On receiving the certificate, client validates it. Only after a successful validation should a secure connection should be established.

Certificate Authorities. Digital certificates are created and cryptographically signed by trusted Certificate Authorities (CAs). In order for a client to trust a certificate presented by a server, it is essential for the two entities to trust the same CA. This concept is known as Public Key Infrastructure (PKI). Root CAs may issue signing certificates to other entities, given them the ability to issue more certificates. These entities are known as intermediate CAs. Typically, Operating Systems (OSs) and web browsers come preconfigured with a list of trusted certificates from well known root CAs.

The PKI system forms a hierarchy of trust that begins at Root CAs, and extends outward through intermediate CAs to leaf certificates that are issued to businesses and individuals. The chain of trust starts with a self-signed root certificate which is trusted by all entities. These root certificates are extremely precious for validating other certificates and thus they are handled in an extremely secure and isolated environment. The private keys of root certificates are not kept online to prevent theft, and thus it is relatively rare for root certificates to sign new certificates.

Since root certificate cannot be used to sign all new the certificates, the certificate signing ability is distributed. Root CAs sign and issue intermediate CA certificates which are also capable of signing more certificates. These intermediate CAs may issue leaf certificates, as well as generate further nested levels

of intermediate CA certificates. A leaf certificates are the end-point in the certificate hierarchy, and are presented to clients when they initiate SSL/TLS sessions with a server. Leaf certificates represent the digital identity of a subject (*e.g.* a business or a person), but they are not capable of signing more certificates.

To request a new certificate, the requester first generates a Certificate Signing Request (CSR). The CSR is then sent to a root or intermediate CA. If the CA accepts the CSR, they use it to generate a new certificate and sign it using their private key. The new certificate is then sent back to the requester.

Certificate Format. Technically, an X.509 digital certificate is a file containing various fields. IETF RFC 5280 [22] defines the following fields in a certificate:

- Version
- Serial Number: a unique identifier for each certificate issued by the CA
- Algorithm ID
- Issuer: the identity of the CA that signed the certificate
- Validity: captures the period of time during which this certificate is valid
- Subject: the identity of the entity that owns this certificate, typically a DNS name
- Subject Public Key Info: the subject's public key and the algorithm used to generate it
- Certificate Signature Algorithm: the algorithm used to sign the certificate
- Certificate Signature: the CAs digital signature of this certificate

In addition to these basic fields, the X.509 specification has been extended over the years to include more information. There are several fields that are important to the modern day PKI, including:

- a URL pointing to the location of the Certificate Revocation List (CRL) for this certificate
- Online Certificate Status Protocol (OCSP) information for this certificate
- An OID that is used to identify Extended Validation (EV) certificates

We describe the meaning of these additional fields in the following two sections.

2.2 Extended Validation (EV) certificates

The most common type of certificate on the Internet is a Domain Validation (DV) certificate. When a CA issues a DV certificate, all the CA needs to do is verify that person acquiring the certificate controls a particular domain name. In other words, DV certs bind a private key to a domain name.

However, a second type of certificate exists called an Extended Validation (EV) certificate. An EV certificate binds a domain name to a real world entity, like a human being or a company. A CA will issue an EV certificate to an entity only after a thorough background check has been completed. Thus, EV certificates indicate to clients that they can have additional trust in the identity of the server, although EV certificates do not actually do anything to improve security as a technical level.

As noted above, EV certificates have additional fields which include detailed information about the owner. Furthermore, EV certificates must be signed by CAs which qualify for signing EV certificates. These CAs are identified by a unique number, called OID. While signing an EV CSR, these CAs insert their OID number into the certificate before signing it. These unique OIDs are hard-coded into the web browsers. When browsers receive certificates with these OIDs, they know they have been sent an EV certificate.

2.3 Certificate Revocation

Digital certificates are issued for a specific period of time: they must be periodically renewed before they expire. However, a situation can arise where the certificate has to be invalidated before its expiry date. Potential scenarios include private key compromise or using a weak algorithm to generate the keys. If the certificate owner believes its private key is compromised, it should revoke its certificate as soon as possible. If the owner does not revoke a certificate, the server's identity can be impersonated by an attacker, and private communications can be eavesdropped.

Certificate owners may revoke their certificates at any time by contacting the CA that issued the certificate. Typically, the certificate owner will supply a reason why they are revoking the certificate. It then becomes the CAs responsibility to publish this revocation so that clients can learn that this certificate is no longer valid.

The process of publishing certificate revocations is implemented by two protocols: Certificate Revocation List (CRL) or Online Certificate Status Protocol (OCSP). CRLs are the original mechanism for revoking certificates [22]. In this system, CAs publish lists of serial numbers for certificates that they issued which have been revoked. Certificates issued by this CA contain a publicly available URL called a CRL distribution points where the CRL can be downloaded. When a client receives a certificate, it is their responsibility to download the associated CRL and verify that the certificate has not been revoked.

Although CRLs are a fairly simple mechanism, this system has a significant drawback. Specifically, CRL lists can grow to become quite large, on the order of megabytes. Downloading these large files adds significant slowdown to the establishment of encrypted connections. Furthermore, CRLs may be updated frequently, which means that it is not safe for clients to cache these files for long periods. Thus, clients are stuck between having fast connections, but potentially stale CRLs, or slow connections that are more secure.

OCSP. The OCSP protocol was designed to address issues with the CRL revocation checking mechanism [23]. An OCSP server is an online database of serial numbers for revoked certificates that includes a simple query API. Rather than host CRLs, CAs may instead setup an OCSP server and include its URL in issued certificates. Clients that support OCSP can then query the database to see if a given certificate has been revoked, instead of downloading the entire CRL file.

An enhancement to OCSP called "OCSP stapling" further reduces the overhead for clients. In this scheme, the certificate owner periodically requests the revocation status of their own certificate from the OCSP server. The server returns a signed response with an expiration time. This signed OCSP response can then be "stapled" to the certificate, *i.e.* transmitted to clients. Thus, when a client start a new SSL/TLS connection, they immediately receive the server's certificate as well as a signed attestation that the certificate is valid (until the stapled OCSP response expires).

Initially, OCSP was designed to provide one response at a time. However, since most certificates today are part of a certificate hierarchy, multiple responses regarding the status are needed validate the whole chain. RFC 6961 was introduced in June 2013 which proposes the "Multiple Certificate Status Request" extension to support multiple OCSP responses to a single request [25]. Thus, OCSP is actively evolving towards a more efficient solution for revocation checking. However, to date some browsers still do not support OCSP, and there are many legacy certificates still in use that rely on CRLs for revocation checking instead of OCSP.

Revocation Server Availability. The final issue with revocation checking is that the server specified in the certificate (CRL or OCSP) may be unavailable. In this case, the client has no ability to check whether the certificate has been revoked or not. This is a particularly vexing situation, since it means the availability of a third-party service can impact the availability of a first-party. For example, if a web browser cannot validate the certificate served by a website, it may refuse to allow the user to visit that website, even though the website is online. The "correct" way to deal with this situation is a topic of debate, since certificate owners are very reluctant to allow the availability of third-party services to directly impact their businesses.

2.4 Client-side Certificate Validation

Clearly, certificates and the associated PKI are complicated mechanisms. On one hand, the security of this system relies on clients to thoroughly check the validity of certificates. On the other hand, validation checking is complicated, and there is **no standard** or test suite that specifies exactly how it should be carried out.

Validation of certificates should include checking the expiration date, the validity of the issuer(s), matching the subject name to the hostname serving the certificate, and revocation checking. After validating the fields of the given certificate, the client should validate the entire certificate chain all the way down to a trusted root. While most of these checks can be performed by analyzing the certificate fields locally, revocation checks require querying information from remote servers. Other validation checks are also possible, such as enforcing a minimum bit-length for encryption keys and checking for the use of vulnerable encryption algorithms.

In order to perform revocation checks, certificate fields can be used to identify the technique (CRL or OCSP) and URL path to the server. In case of CRL revocation check, when the client encounters the CRL Distribution Point in the certificate, it should download the CRL from the location and verify the serial number against the list. In the case of OCSP, the client should query for the revocation status of the certificate from the specified OCSP server. Stapled OCSP responses are the best-case scenario, although the revocation status of associated intermediate and root certificates may still need to be checked explicitly.

On successful validation of a certificate, the client can finish establishing a secure connection to the server. Typically, web browsers indicate this visually in the address bar. For example, Internet Explorer and Firefox display a grey lock icon, while Google Chrome displays a green lock. In case of an EV certificate, Google Chrome and Firefox display the certificate owner's name in a green box in the left corner of address bar, while Internet Explorer turns the entire address bar green and prints the certificate owner's name in the right corner of the address bar.

In cases where certificate validation fails, clients should display some sort of error or warning to the user. Depending on the severity of the problem with the certificate, the user may be allowed to override the error and proceed, or they may be totally blocked from continuing the connection.

Chapter 3

Methodology

In this study our goal is to evaluate whether modern browsers implement correct and thorough validation of certificates. In this section, we describe the setup for experiments. *First*, we identify the subset of browsers, versions, and platforms that we will analyze. *Second*, we define the types of invalid certificates we will use to test browsers, and describe the process for generating these certificates. *Third*, we discuss the configuration of the web servers that will host our certificates. *Lastly*, we perform validation tests on our experimental platform to ensure that it correctly serves valid certificate chains and CRLs.

3.1 Goals and Methods

The goal of this study is to examine the certificate validation approaches used by modern browsers. In theory, all modern web browsers should perform stringent validation checks on certificates before establishing a secure connection. However, as discussed in § 2.4, there is no standard that specifies how clients should validate certificates. This ambiguity leaves the task of implementing certificate validation up to browser developers, who may or may not follow best-practices in their implementations. Furthermore, certificate validation is non-trivial, and involves many checks on data from multiple certificates and network services. These technical challenges open up the possibility that browsers may have buggy or incomplete certificate validation code.

To study the certificate validation behavior of browsers, there are several concrete steps we must take, which we will elaborate on in the next four sections. *First*, we identified a set of test cases that cover typical certificate pathologies that browsers should be able to identify and respond to. These test cases are shown in Table 3.1, and include issues intrinsic to certificates (*e.g.* expiration), revocation, and network-related issues (*e.g.* CRL unavailable). As shown in the columns of Table 3.1, the test cases cover pathologies in the leaf certificate, the CA signing certificate, and the root certificate. Certificate chains of length three are sufficient to examine issues on all major classes of certificates, and thus we do not examine longer chains. All of our experiments were conducted using standard and EV certificates.

We expect that browsers will be able to successfully identify many of the pathologies in Table 3.1, and warn the user or show an error message. This includes expired certificates, self-signed non-root certificates, and invalid certificates. To create invalid certificates, we generated certificates for a common name that did not match the DNS name of the host serving the certificate. Similarly, we expect that browsers will show an error if the root certificate for a given chain is unavailable, since this would prevent the chain from validating. The more interesting test cases are related to revocations: anecdotal evidence suggests that many browsers fail to check for revocations, or silently accept certificates even when the CRL is unavailable and the revocation status cannot be checked.

Second, we must select the target browsers for our study. Given that there are many web browsers on many different platforms (and multiple versions of each browser), it is essential to narrow down the field. In § 3.2, we discuss how we selected browsers for our study, based on popularity metrics as well

Leaf Server Certificate	Intermediate CA Certificate	Root CA Certificate
Expired	Expired	Expired
Invalid	Invalid	-
Self Signed	Self Signed	-
-	-	Certificate Unavailable
Certificate Revoked	Certificate Revoked	-
CRL Unavailable	CRL Unavailable	-

Table 3.1: Test cases used in this study to examine the certificate validation behavior of browsers. All tests are conducted using EV and non-EV certificates.

as practical considerations. We also discuss the test environment we hosted the browsers in, to control for effects like caching.

Third, to implement the test cases in Table 3.1, we require many different types of certificates and CRLs. We give an overview of the certificate generation process used in this study in § 3.3. *Fourth*, to analyze the behavior of the target browsers, the certificates from stage three must be hosted by websites. In 3.4, we describe our experimental testbed, including the configuration of the intermediate CA and leaf servers that host the certificates generated in stage three. *Fifth* and finally, in § 3.5, we validate the correctness our testbed using a known-good environment (*i.e.* correct certificates, valid CRLs, and a browser that performs stringent certificate validation) to ensure that the certificate chain is valid and accessible.

3.2 Selecting Browsers and Platforms

Selecting browsers is the first step for our study. While selecting browsers, there were several challenges to address. *First,* there are many web browsers in the Internet space with many versions of each. Each browser is implemented differently in different platforms. Clearly, there are too many combinations of browsers, versions, and platform for us to evaluate them all. *Second,* the availability of the source code impacts our ability to evaluate and understand the behavior of each browser. We preferred open source browsers for our study.

We used real-world usage data to guide our browser selection process. Data from Netmarketshare [13] from January 1, 2014 to September 30th, 2014 shows that Internet Explorer dominates the browser market with 58.17% usage. Google Chrome (18.48%) and Mozilla Firefox (16.36%) are the next leaders in the browser space.

While the statistics show Internet Explorer is the most common browser, we have selected Google Chrome and Mozilla Firefox for this study. First, since Internet Explorer is not open source, it may be challenging to identify the root causes of anomalous certificate validation behavior in our experiments. Second, it turns out that Internet Explorer has extensive revocation checking practice [4]. As we will show in § 3.5, Internet Explorer's certificate validation procedure is very complete, and thus we treat it as the gold standard for validation purposes.

In § 4, we examine the certificate validation behavior of Google Chrome and Mozilla Firefox. Both browsers are open source and therefore we are able look into the source code and explain the root causes that lead to our test results.

To narrow down the evaluation, we selected Google Chrome version 39 and Mozilla Firefox version 38 for our experiments. Data from Netmarketshare [13] shows a significant gain in popularity of theses versions of the browsers since Aug 2014. This behavior makes sense given that these are the latest versions of the respective browsers, and both browsers urge users to automatically update to the latest version. Hence, we selected the most popular versions of the browsers during the time of our study, which are the current versions as well.

Browser behavior may be impacted by the underlying platform. To study browser behavior on different OS we selected Windows 7, Ubuntu 14.04, and Android 4.4.2 as test platforms. We selected

Browser	Version	Platform
Google Chrome	39	Windows 7 Ubuntu 14.04 Android 4.4.2
Mozilla Firefox	38	Windows 7 Ubuntu 14.04
	33.1	Android 4.4.2

Table 3.2: Browser versions studies in the platforms

Windows 7 and Ubuntu 14.04 to analyze desktop versions of the web browsers as these are the most widely used platforms. As mobile browsing is gaining popularity, we selected the most current Android version 4.4.2 to test our browsers. Table 3.2 illustrates the selected browsers, versions and platform for our study.

3.3 Generating Certificates and CRLs

In order to run the test cases shown in Table 3.1, we had to generate various certificates. For our experiments, we generated a certificate hierarchy of depth 3, *i.e.* we play the role of a root CA, an intermediate CA, and a customer with a leaf certificate. Thus, for each experiment, we generated 1) a self-signed root CA certificate, 2) an intermediate CA signing certificate issued by the root CA, and 3) a leaf certificate issued by intermediate CA. We also generated CRLs associated with the leaf and CA certificates.

We leveraged the standard OpenSSL version 1.0.1 [16] utility to generate certificates. OpenSSL is an open source, full featured toolkit that is able to generate, sign, and validate certificates, as well as implement the SSL/TLS network protocols and associated encryption routines. We chose OpenSSL because it is the most widely used SSL/TLS suite [15]. Next, we discuss the steps we took to generate certificates, keys, and CRLs for our experiments.

Step 1: Configuring OpenSSL. The configuration file used by OpenSSL when generating certificates is `openssl.cnf`, which controls the contents of generated X.509 certificates. In our case, we created four different versions of this file, corresponding to EV and non-EV intermediate CA and root certificates. In the examples below, we store files related to the root CA in the `/etc/apache/ssl/CA/` folder, while files for the intermediate CA are stored in `/etc/apache/ssl/`.

This setup allows us to quickly generate different versions of these certificates with varying properties (*e.g.* expiration dates). Each configuration was configured with a unique `CA_default` working directory (which must be specified as an absolute path).

We wrote two "extensions" in the OpenSSL configuration files to control the signing capabilities of generated certificates. The extension `usr_cert` defines that the output will be a leaf certificate that cannot sign more certificates. In contrast, the extension `v3_ca` defines that the output is a signing certificate. The definitions used in our experiments are given below:

```
[ usr_cert ]
basicConstraints=CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
crlDistributionPoint =  URI://path-to-crl

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = CA:true
keyUsage = cRLSign, keyCertSign
```

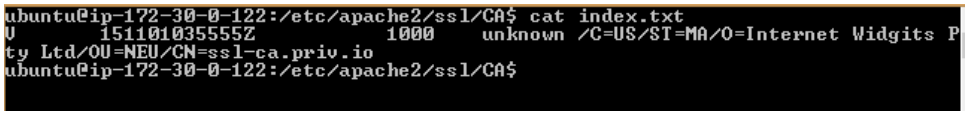

Figure 3.1: Index.txt file for our root CA after generating one intermediate CA certificate.

```
crlDistributionPoint =  URI://path-to-crl
```

Note that the `crlDistributionPoint` variable must be modified to point to the URL hosting the CRL associated with each certificate. In our test scenario, we hosted the CRL for root CA on `http://ssl-ca.priv.io/crl1.crl` and `http://ssl-ca.priv.io/crl2.crl` for the intermediate CA.

OpenSSL includes many other configuration variables that we have not discussed here. All unspecified variables have been left at their defaults values.

Step 2: Generating Root Keys and Certificates. Using the configurations defined in step 1, we next move on to generating a public-private keypair and associated certificate for our root CA. Specifically, we use the following commands to generate the root keypair and certificate:

```
$ openssl genrsa -aes256 -out /etc/apache/ssl/CA/root.key 4096
$ openssl req -new -x509 -days 3 -key /etc/apache/ssl/CA/root.key -sha256 -extensions
     v3_ca -out /etc/apache/ssl/CA/root.crt
```

The certificate for the root CA is self-signed. As discussed above, the option `-extension v3_ca` allows the generated certificate to sign more certificates. Option `-days 3` defines the certificate expiry.

In addition to the keypair and certificate, other information is also maintained for the root CA. Specifically, the file `index.txt` contains the database of the certificates signed by the root CA. Figure 3.1 shows an example of the `index.txt` file after the root generated one intermediate CA certificate. The `index.txt` file is a tab separated file containing six fields: status, expiration date, revoked date, serial number, file name of the certificate, and subject name. The `serial` file contains the serial number for the next certificate to be signed, which is incremented each time a certificate is signed.

Step 3: Generating Intermediate CA Keys and Certificates. The process for generating the intermediate CA certificate is different than for the root certificate, since the intermediate certificate must be signed by the root. Thus, rather than generate a keypair and a self-signed certificate, instead we use OpenSSL to generate a keypair and a Certificate Signing Request (CSR) using the following commands:

```
$ openssl genrsa -aes256 -out /etc/apache/ssl/ca-int.key 4096
$ openssl req -config /etc/apache/ssl/openssl.cnf -sha256 -new -key /etc/apache/ssl/
    ca-int.key -out /etc/apache/ssl/ca-int.csr
```

The CSR includes the subject name and contact details of our intermediate CA. The CSR is then sent to the root CA which will sign it and return it to the intermediate CA. The command to sign the CSR is:

```
$ openssl ca -config /etc/apache/ssl/CA/openssl.cnf -keyfile /etc/apache/ssl/CA/root.
    key -cert /etc/apache/ssl/CA/root.crt -extensions v3_ca -notext -md sha256 -in /
    etc/apache/ssl/ca-int.csr -out /etc/apache/ssl/ca-int.crt
```

Since the intermediate CA will be required to sign leaf certificates, the root CA will sign the intermediate CA with the option `-extension v3_ca`. Furthermore, the root's `index.txt` is updated to add the record of the newly signed certificate. `index.txt` and `serial` files are also created for the intermediate CA (separate from the files for the root), since the intermediate CA will be signing leaf certificates. Finally, the correctness of the generated intermediate CA certificate can be verified using the command:

```
$ openssl ca -keyfile /etc/apache/ssl/CA/root.key -cert /etc/apache/ssl/CA/root.crt -
    verify /etc/apache/ssl/ca-int.crt
```

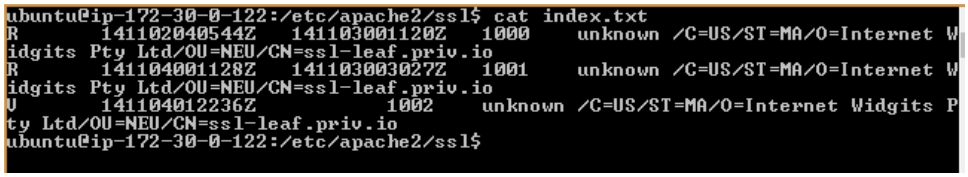

Figure 3.2: Intermediate CA `index.txt` file with a revoked certificate.

Step 4: Generating the Leaf Keys and Certificates. The process for generating leaf certificates is similar to generating intermediate certificates. As shown in the following example, we generate a keypair and a CSR for the leaf certificate:

```
$ openssl genrsa -out /etc/apache/ssl/newcerts/server.key 4096
$ openssl req -config /etc/apache/ssl/openssl.cnf -sha256 -new -key /etc/apache/ssl/
    newcerts/server.key -out /etc/apache/ssl/newcerts/server.csr
```

The CSR is sent to the intermediate CA for signing and generating the certificate. The command to generate the signed digital certificate is:

```
$ openssl ca -keyfile /etc/apache/ssl/ca-int.key -cert /etc/apache/ssl/ca-int.crt -
    extensions usr\_cert -days 1 -notext -md sha256 -in /etc/apache/ssl/newcerts/
    server.csr -out /etc/apache/ssl/newcerts/server.crt
```

Unlike the intermediate certificate, leaf certificates may not sign additional certificates. Thus, the intermediate CA signs the leaf server with option `-extensions usr_cert`. The leaf certificate can be verified using the public keys associated with the intermediate and root certificates.

Step 5: Revoking Certificates and Generating CRLs. In several of the experiments outlined in Table 3.1, we will revoke certificates that we have previously generated. We use the following OpenSSL command to revoke a certificate (a leaf certificate in this case), which changes the target certificates descriptor from "V" to "R" in the `index.txt` file as shown in 3.2

```
$ openssl ca  -config /etc/apache/ssl/openssl.cnf -keyfile /etc/apache/ssl/ca-int.key
    -cert /etc/apache/ssl/ca-int.crt -revoke /etc/apache/ssl/newcerts/server.crt
```

In addition to revoking certificates, we must publicize the revocations by generating CRLs and hosting them at the URL specified in the `openssl.cnf` file. We use the following OpenSSL command to generate CRLs:

```
$ openssl ca -keyfile /etc/apache/ssl/ca-int.key -cert /etc/apache/ssl/ca-int.crt -
    config /etc/apache/ssl/openssl.cnf -gencrl -out /var/www/html/crl2.crl
```

Addendum: Generating EV Certificates. The five steps outlined about allow us to generate certificate chains, as well as revoke certificates within the chain. However, thus far our discussion has focused on vanilla certificates; to generate EV certificates, several of the above steps must be modified.

Prior to generating an EV certificate, `openssl.cnf` must be modified. Most importantly, the issuing certificate authority has to be identified as capable of signing EV certificates. This is achieved by inserting a special OID value into the certificate. As discussed in § 2.2, these special OID numbers are announced by major CAs, and hard-coded into browsers so that they can identify EV certificates. For this study, we borrowed the OID from an existing CA (Digicert [14]) and inserted it into our `openssl.cnf`. Alternatively, we could have generated a random, fresh OID and inserted it into the source code of the target browsers, but this would require significantly more work than borrowing an existing OID.

As shown in the example below, our `openssl-ev.cnf` file contains the borrowed OID, as well as several other fields that provide details about the issuer. Note that the `v3_ca` and `usr_cert` extensions shown below are abridged: they contain all the commands shown above in addition to what is shown in this example.

```
[ new_oid ]
businessCategory=2.5.4.15
streetAddress=2.5.4.9
stateOrProvinceName=2.5.4.8
countryName=2.5.4.6
jurisdictionOfIncorporationStateOrProvinceName=1.3.6.1.4.1.311.60.2.1.2
jurisdictionOfIncorporationLocalityName=1.3.6.1.4.1.311.60.2.1.1
jurisdictionOfIncorporationCountryName=1.3.6.1.4.1.311.60.2.1.3

[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
businessCategory = supplied
jurisdictionOfIncorporationCountryName = supplied
jurisdictionOfIncorporationStateOrProvinceName = supplied
jurisdictionOfIncorporationLocalityName = supplied

[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
businessCategory = supplied
jurisdictionOfIncorporationCountryName = supplied
jurisdictionOfIncorporationStateOrProvinceName = supplied
jurisdictionOfIncorporationLocalityName = supplied

[ req_distinguished_name ]
# OID 2.5.4.15
businessCategory = Business Category (For example, 'V1.0, Clause 5.(c)')
# OID 1.3.6.1.4.1.311.60.2.1.1
jurisdictionOfIncorporationLocalityName = Inc. Locality
jurisdictionOfIncorporationStateOrProvinceName = Inc. State/Province
jurisdictionOfIncorporationCountryName = Inc. Country

[ v3_ca ]
certificatePolicies = 2.16.840.1.114412.2.1
...

[ usr_cert ]
certificatePolicies = 2.16.840.1.114412.2.1
...
```

After the configuration of the `opensslev.cnf` file, EV root certificates can be generated using the steps outlined above. CSRs are also generated the same as above, except the `opensslev.cnf` is used. Once the CSR is generated, it is signed using the policy option with value `policy_anything`. This option is used to accept any fields in the subject DN which is mandatory. Hence it allows the certificate to have values in the certificate that are different than those from the issuer certificate. In absence of this option, policy defaults to *policy_match* which put restrictions on the subject name matching [18].

In summary, the modified commands for generating an EV leaf keypair, CSR, and certificate are:

```
$ openssl genrsa -out /etc/apache2/ssl/newcerts/server-ev.key 4096
$ openssl req -config /etc/apache2/ssl/openssl_ev.cnf -sha256 -new -key /etc/apache2/
    ssl/newcerts/server-ev.key -out /etc/apache2/ssl/newcerts/server-ev.csr
$ openssl ca -keyfile /etc/apache2/ssl/ca-int.key -cert /etc/apache2/ssl/ca-int.crt -
    config /etc/apache2/ssl/openssl_ev.cnf -extensions usr_cert -policy
    policy_anything  -days 2 -notext -md sha256 -in /etc/apache2/ssl/newcerts/server-
    ev.csr -out /etc/apache2/ssl/newcerts/server-ev.crt
```

3.4 Experimental Testbed

The next step in our methodology is setting up a test environment for the browsers, and web servers to host our certificates. To test Chrome and Firefox on the desktop, we installed them in virtualized instances of Windows 7 and Ubuntu 14.04. We leveraged Wireshark to record network traces from the desktop browsers. To conduct experiments on mobile devices, we installed Chrome and Firefox on an Android 4.4.2 device from the Play Store. To capture browser traffic on Android, we leveraged an application from the Play Store called tPacketCapture, which captures packet traces by emulating a VPN. Packet captures generated by tPacketCapture are in PCAP format, and are thus compatible with Wireshark.

For this study, we required a minimum of two websites supporting HTTPS: one functioning as the intermediate CA, and the other functioning as the leaf server. Although it is possible to setup two SSL-enabled websites on the same host using the SSL Server Name Indication (SNI) extension, we opted to go the more traditional route and use separate hosts for each server. Each of our servers was a virtual machine rented from Amazon Web Services (AWS) running Ubuntu Server 14.04 with Apache2 and OpenSSL installed. We configured two subdomains of a DNS entry under our control [30] to point to the two servers: `ssl-ca.priv.io` resolved to the IP address of the intermediate CA server, while `ssl-leaf.priv.io` resolved to the leaf server.

Configuring the Intermediate CA Server. We configured the Apache instance on the intermediate CA server by modifying the `default-ssl.conf` file as shown below:

```
ServerName ssl-ca.priv.io:443
SSLEngine on
SSLCertificateFile /etc/apache2/ssl/ca-int.key
SSLCertificateKeyFile /etc/apache2/ssl/ca-int.crt
```

This configuration is straightforward: it specifies the location of the intermediate CA's private key and the corresponding certificate.

Configuring the Leaf Server. The configuration of Apache on the leaf server is slightly more complicated, as shown in the following example snippet from `default-ssl.conf`:

```
ServerName ssl-leaf.priv.io:443
SSLEngine on
SSLCertificateFile /etc/apache2/ssl/server.key
SSLCertificateKeyFile /ec/apache2/ssl/server.crt
SSLCACertificateFile /etc/apache2/ssl/ca-int.crt
SSLCertificateChainFile /etc/apache2/ssl/ca-chain.crt
```

In addition to the leaf's private key and corresponding certificate, the configuration also points to a copy of the intermediate CA's certificate, as well as a certificate chain file. The chain file is simply the concatenation of the root and intermediate CA's certificates.

Installing Root Certificates. To complete the experimental testbed configuration, the root certificates must be installed on the client machines. On Windows, Chrome imports root certificates from the Windows OS Certificate Store, thus we installed our root certificates into this store. On Linux, Google Chrome has its own repository of the certificates. Thus, we imported the root CA certificate directly into the browser. On Android, Chrome retrieves root certificates from the OS, which is managed through the Play Store. On the other hand, Firefox has its own repository of root certificates, hence we imported the root certificate directly into Firefox on all three target platforms.

3.5 Validation

The last step in our methodology is to validate that our testbed is correct. To validate our set up, we need to check two factors. *First,* the certificate hierarchy is established and validated by the browser. *Second,* the CRLs which are hosted on a public network are accessible by the browser. Validation of the testbed

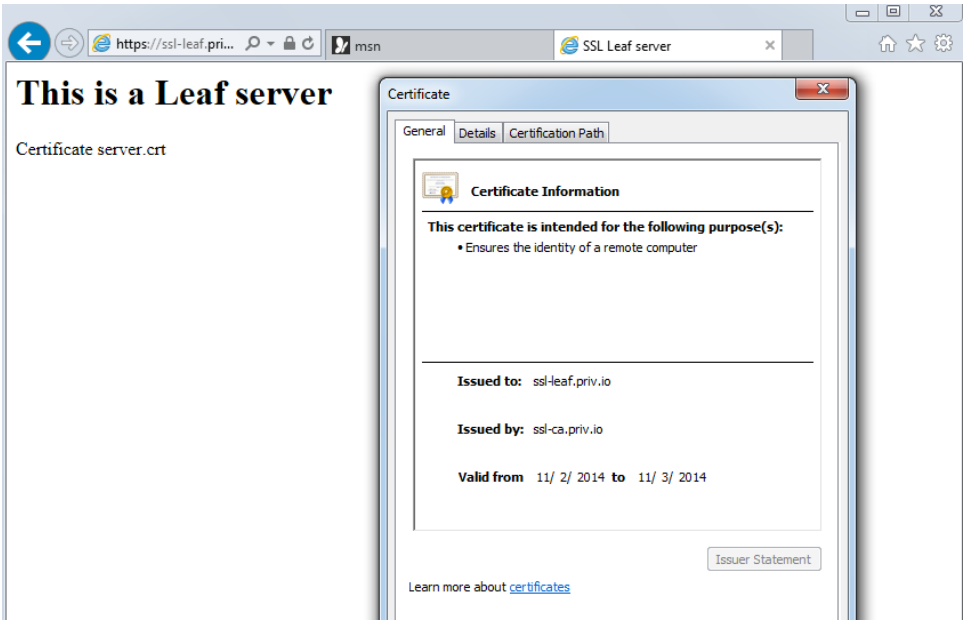

Figure 3.3: Leaf server successfully loaded in the Internet Explorer.

is critical: if any of these basic checks fails, it means that we cannot reliably evaluate the behavior our target browsers.

To validate our test-bed, we used Internet Explorer 8 running on Windows 7 to visit `https://ssl-leaf.priv.io`. For this experiment, we generated a valid chain of non-expired, non-revoked certificates and installed them on the two servers. We chose Internet Explorer for this experiment because the documentation for the browser claims that it performs a revocation check on *all* downloaded certificates. Specifically, the stated behavior is that Internet Explorer will attempt an OCSP check on each certificate, but fall back to a CRL check if OCSP is not enabled on the certificate [4]. This functionality is described in the specification for the Windows cryptography API, which Internet Explorer relies on [9].

We verified whether Internet Explorer actually performs these revocation checks, and simultaneously validated our testbed, by using the browser to visit `https://ssl-leaf.priv.io`. We cleared the Windows CRL cache and installed our root certificate before conducting this experiment. We used Wireshark to capture packet traces during the experiment. As shown in Figure 3.3, Internet Explorer was able to successfully load the root page from the leaf server without generating any SSL/TLS errors or warnings. This demonstrates that our testbed is capable of serving a leaf and intermediate CA certificates, and that the chain was successfully validated by the browser.

Furthermore, the Wireshark traces reveal that Internet Explorer did indeed request and receive the CRLs corresponding to the leaf and the intermediate certificates. Figure 3.4 shows a portion of the Wireshark trace, corresponding to a CRL request to `ssl-ca.priv.io`. This confirms that our testbed is able to serve CRLs corresponding to the given leaf and intermediate certificates.

As a final check, we examine the CRL cache that is maintained in the OS by the Windows cryptography API. As shown in Figure 3.5, the CRLs for our servers were placed in the cache during the experiment. This acts as final confirmation that the CRLs were successfully retrieved and parsed by Internet Explorer.

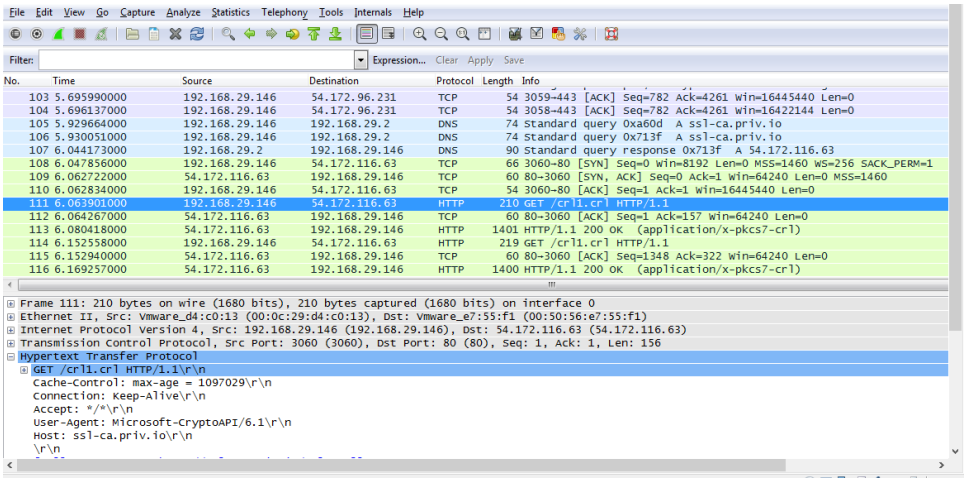

Figure 3.4: Internet Explorer makes a CRL requests to ssl-ca.priv.io.

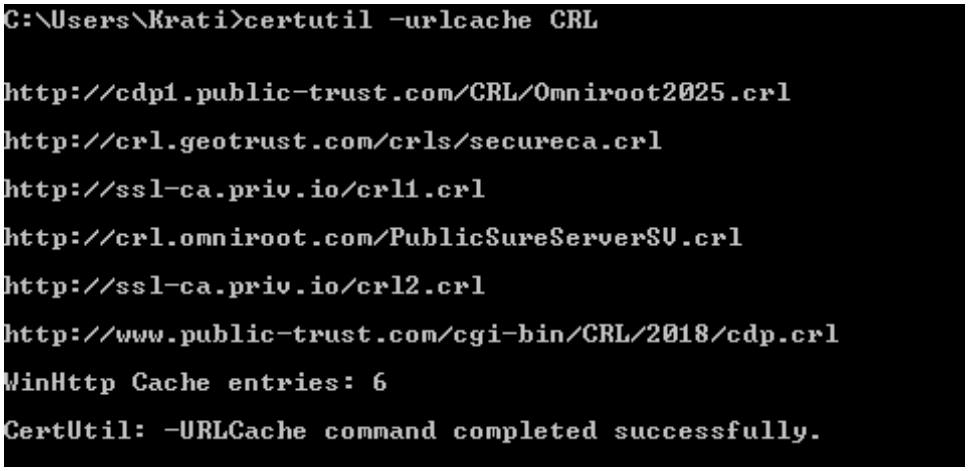

Figure 3.5: Both CRLs from our testbed are cached in Windows after the validation experiment.

Chapter 4

Analysis

The purpose of this study is to analyze the approaches adopted by modern browsers during certificate validation. To achieve this goal, we performed various tests on the browsers specified in Table 3.1. The results of our experiments are summarized in Table 4.1: checkmarks (✓) indicate that a browser exhibited correct behavior, while X denotes cases where the browser failed to exhibit the correct behavior (X* indicates a partial failure, which we explain below). Dashes indicate tests that could not be run for technical reasons. Note that, since each test involves certificates with some kind of error, "correct behavior" means showing an error or warning to the user. Conversely, browsers fail by not detecting the certificate error and allowing the website to load. Figure 4.1 shows an example of Chrome loading the homepage from the leaf server, which ideally should not occur during any of the tests specified in Table 4.1.

As we can see from Table 4.1, Chrome and Firefox exhibit a range of different correct and incorrect behaviors. In the case of Chrome, the behavior of the browser is platform dependent, while Firefox's behavior appears to be platform independent. In the following sections, we analyze the results of each browser in detail.

4.1 Google Chrome

In this section, we examine Chrome's certificate validation approach. As shown in Table 4.1, Chrome's behavior varies depending on the underlying OS. This is because Chrome is designed to use the Windows cryptographic API and Android APIs on those platforms, while it uses the Mozilla Network Security Services (NSS) library on Linux. Given this variability, we structure our discussion of Chrome around specific types of certificate errors.

Expired, Invalid, and Unavailable Certificates. For the most part, Chrome correctly displays errors when presented with expired certificates, invalid certificates, and cases where the root certificate

Browser	Platform	Expired			Invalid		Self Signed		Root Cert.	Revoked (Std.)		Revoked (EV)	CRL Unavail.	
		L	I	R	L	I	L	I	Unavail.	L	I	L	L	I
Chrome	Win. 7	✓	✓	✓	✓	✓	X	X	✓	X*	X*	✓	X	X
	Ubuntu	✓	✓	✓	✓	✓	✓	✓	✓	X	X	X	X	X
	Android	✓	✓	X	✓	✓	X	X	X	X	X	X	X	X
Firefox	Win. 7	✓	✓	✓	✓	✓	✓	✓	✓	X	X	X	X	X
	Ubuntu	✓	✓	✓	✓	✓	✓	✓	✓	X	X	X	X	X
	Android	✓	-	-	✓	✓	✓	✓	-	X	X	X	X	X

Table 4.1: Test cases used in this study to examine the certificate validation behavior of browsers. All tests were conducted using EV and non-EV certificates. "L", "I", and "R" indicate tests on the leaf, intermediate CA, and root certificates, respectively.

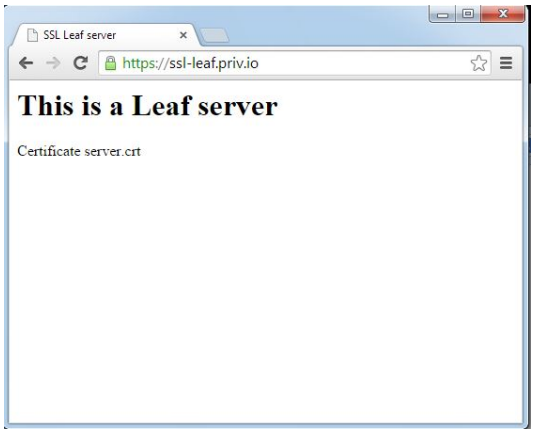

Figure 4.1: Google Chrome loading the website

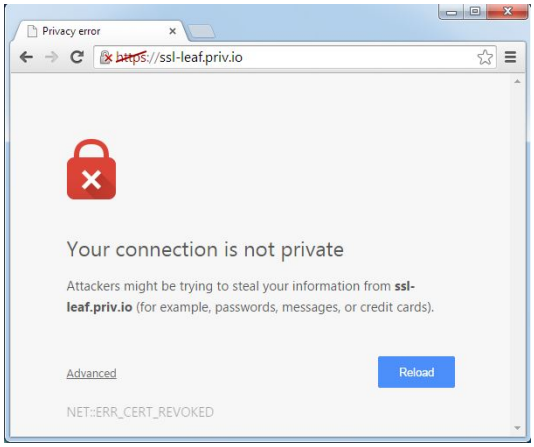

Figure 4.2: Google Chrome displaying error to user along with blocking access to the site.

was not installed. Figure 4.2 shows an example of this error page.

The notable exceptions to this correct behavior occur on Android. As shown in Table 4.1, Chrome on Android does not display an error if the root certificate is expired or unavailable. When the root certificate is missing, the certificate details dialog in Chrome displays the details of the leaf and intermediate certificates, but shows no information on the root certificate. Both of these behaviors are very problematic from a security perspective, and likely represent bugs in the Android cryptographic API.

Self Signed Certificates. When presented with a self signed (non-root) certificate, browsers should display an error or warning to the user. Self signed certificates do not belong to a chain rooted in a trusted CA, thus they should not be trusted by default. Chrome on Linux exhibited correct behavior by displaying a warning to the user when presented with a self signed certificate.

However, Chrome on Windows and Linux does not exhibit correct behavior in this test. On these platforms, Chrome silently accepts the self signed certificate and loads the site. The "green padlock" icon is shown in the URL bar, which indicates to the user that this connection is secure (which, in this case, is not true). Given that this error is platform dependent, it is clear that Chrome is not using the system APIs properly on Windows and Android.

Revoked Certificates. The Chrome documentation states that Chrome does not check CRLs for revocation information. Instead, Chrome uses a customized implementation of CRL revocation checking known as CRLsets [8]. A CRLset is simply a file that is compiled and distributed by Google that contains a list of revoked certificates. Chrome periodically downloads updated versions of this file. However, the total size of the CRLset is roughly 250KB, which means it must only contain a subset of all revoked certificates. How and from where these revocations are selected remains a mystery.

As shown in Table 4.1, Chrome failed almost all of our tests involving revoked certificates. Obviously, there is no way for Google to know about the certificates used in our experiments, so they do not appear in the CRLset. Since Chrome does not download CRLs (with two exceptions, discussed below), it is unable to detect when our certificates were revoked. Furthermore, since Chrome does not check CRLs, it did not notice when our CRLs were unavailable.

There are two exceptions to Chrome's revocation checking behavior, both of which occur on Windows 7. The first exception stems from that fact that Chrome imports CRL information from the Windows cryptographic cache if it present. Thus, if up-to-date CRLs happen to be cached in the crypto library, Chrome correctly identifies the revoked certificate. This observation leads to the two X* entries in Table 4.1: if Internet Explorer is used to view a website, the CRL information will be cached by Windows. If Chrome is then used to view the same website, it will import the correct CRL information and detect the revoked certificate. In the absence of current CRL in the Windows cache, Google Chrome

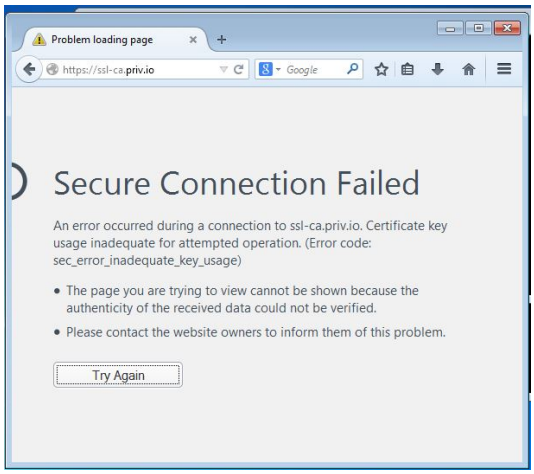

Figure 4.3: Firefox displaying error to user along with blocking access to the site

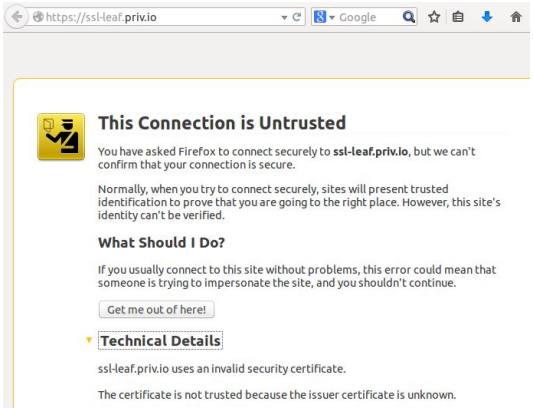

Figure 4.4: Firefox displays a warning to user, while giving an option to continue

is entirely depend on the CRLset.

The second exception is that Chrome will detect revoked EV certificates on Windows 7. According to the Chrome documentation, Chrome does perform a CRL check when presented with an EV certificate. However, in our testing we only observed this happening on Windows. On Linux, Chrome uses the NSS library to validate certificates, and NSS does not support CRLs at all. Furthermore, Android also does not support EV certificates [6]. Thus, although Chrome claims to perform more thorough validation checks on EV certificates, our results show that this claim is incomplete at best.

4.2 Mozilla Firefox

In this section, we analyze Mozilla Firefox's certificate validation approach. Firefox relies on NSS for certificate validation across all platforms (which makes sense, since NSS is also developed by Mozilla). This gives Firefox more uniform and predictable behavior than Chrome, which is evident in Table 4.1.

Firefox successfully identified all of the expired and invalid certificates we directed it to, as well as chains with missing root certificates. In these cases, it displayed the error shown in Figure 4.3. Similarly, Firefox displayed the warning shown in Figure 4.4 when presented with a self signed (non-root) certificate. Unlike Chrome, Firefox exhibited the correct behavior in all of these tests.

Revoked Certificates. However, like Chrome, Firefox failed to identify any revoked certificates during our experiments. Mozilla recently decided to disable all CRL checking in Firefox, instead choosing to rely totally on OCSP checks [3]. Thus, Firefox failed to notice when certificates were revoked, or when the CRLs were unavailable.

Firefox on Android. Firefox on Android is a special case, and thus deserves separate discussion. Firefox on Android comes bundled with a set of trusted root certificates, and we were unable to force it to import the root certificates we generated for our experiments. Thus, we were unable to perform some of our experiments on Firefox for Android (as shown by the dashes in Table 4.1). However, other than this issue, Firefox on Android relies on the same codebase as desktop Firefox, and thus we see that's its behavior is the same as its desktop counterpart in all the tests we ran.

Chapter 5

Conclusion

Digital certificates are used to establish the identity of servers on the Web, and form the building blocks on which secure, encrypted communications are built. Certificates belong to a tree-structured Public Key Infrastructure (PKI), where trust is assumed in Certificate Authorities (CAs) at the root of the tree. CAs are then responsible for issuing certificates to third-parties that bind DNS names to public keys; these leaf certificates are then presented to clients, who can then verify that the certificate chain is valid and trusted.

However, even given a perfect PKI, the security of clients cannot be guaranteed if clients do not perform correct and thorough validation checks on certificates. If a client does not properly validate certificates, it becomes vulnerable to several attacks, including man-in-the-middle attacks that compromise the confidentiality and integrity of communications and spoofing attacks that allow someone to impersonate a trusted third-party. Thus, it is crucial that clients thoroughly validate certificates before trusting remote servers or transmitting data over the Internet.

In this study, we evaluated the certificate validation routines of Google Chrome and Mozilla Firefox on Windows 7, Ubuntu 14.04, and Android 4.4.2 when faced with different types of invalid certificates. These tests include expired, invalid, self-signed, and revoked standard certificates, as well as EV certificates. We built an extensive testbed that allowed us to experiment with different combinations of leaf, intermediate CA, and root certificates.

Alarmingly, we discovered many problems with these browsers' approach to validation. Chrome exhibits different erroneous behavior on different platforms, including accepting expired root certificates on Android, and accepting self-signed certificates with no visible warnings on Windows 7 and Android. Chrome's varied behavior stems from the fact that it relies on Windows and Android APIs to perform certificate validation. On the other hand, Firefox relies on the same certificate validation library on all platforms, which gives it uniform behavior.

In addition the bugs we identified in Chrome, both browsers fail to identify revoked certificates (with only a single exception). In Chrome, this behavior stems from the browser's sole reliance on CRLsets [8], which only contain a small subset of all revoked certificates. In Firefox, Mozilla has decided to disable all CRL checking and rely entirely on OCSP. In both cases, the browser makers' decision to abandon standard CRL checking is problematic, since the alternative implementations are not robust. Now more than ever, it is critical that browsers perform revocation checks on certificates, since revocation checking is the first line of defense against certificates that are stolen during major data breaches like Heartbleed [20, 29].

5.1 Recommendations for Browser Designers

In our study, we confirmed that the revocation checking techniques employed by Google Chrome and Mozilla Firefox are flawed. Although Mozilla's decision to focus on OCSP is progressive, in reality

very few certificates support this newer protocol. This leaves Firefox users vulnerable to man-in-the-middle and spoofing attacks. Chrome's reliance on CRLsets is troubling, since CRLsets only cover a small subset of all revocations, and it is not clear where the data for the CRLset is compiled from, or how quickly it is updated.

Surprisingly, as shown in § 3.5, Internet Explorer implements the most robust revocation check of any browser we examined. This suggests that other browser vendors should also adopt Internet Explorers approach, *i.e.* use OCSP as the primary check but default to a CRL check if OCSP is not present in the certificate.

Furthermore, during our experiments we noticed that Chrome accepts self-signed leaf and intermediate CA certificates in some cases. We suspect that this behavior is designed to please business owners, who sometimes deploy man-in-the-middle proxies that serve self-signed certificates. Although businesses may appreciate this feature, it is clearly detrimental to the overall security of Chrome users, *i.e.* we were trivially able to make Chrome load a webpage with a self-signed certificate and no warnings were shown to the user.

In this case, Chrome should adopt Firefox's behavior, which is to show the user a warning and allow them to make the decision whether to continue. Although security warnings are sometimes confusing to users, this is still the better approach overall, since it does not leave users exposed to self-signed certificates from unscrupulous websites.

5.2 Recommendations for Users

Revocation checking implementation is a choice browser designers make. As we have observed in this research, Google Chrome and Mozilla Firefox's implementation does not perform comprehensive revocation check. Thus, we recommend that users perform security critical tasks in Internet Explorer, since it implements the most thorough revocation checks of any browser we examined.

Perhaps more troubling is the observation that Chrome and Firefox on Android do not perform revocation checks. Smartphones and tablets are rapidly becoming people's primary means of browsing the Web. This behavior includes conducting security-critical tasks like online banking. For the time being, we would advise users not to engage in security-critical web browsing on mobile devices, since 1) mobile browsers do not implement revocation checking, and 2) mobile communications are more likely to be man-in-the-middled than fixed-line communications [5].

5.3 Future Work

Due to time constraints, our study only analyzed the certificate validation approach implemented by Google Chrome and Mozilla Firefox on Windows 7, Ubuntu 14.04, and Android OS. There are several obvious ways to expand the scope of this study, including examining more browsers (Internet Explorer, Safari, and Opera) on more platforms (OSX and iOS).

Additionally, there are more certificate pathologies that should be examined by future work. In particular, we did not examine the ability of Chrome or Firefox to detect OCSP revoked certificates, certificates with invalid stapled OCSP credentials, or cases where the OCSP resolver is unavailable. As OCSP becomes more widely deployed, the behavior and correctness of browsers with regards to this protocol will become more important.

Finally, the composition and dynamics of the Chrome CRLset remain a mystery. It is not clear where Google obtains information about revoked certificates, how they decide which revocations to add to the CRLset, or how timely the additions to the CRLset are. Given that the CRLset is a critical component of Chrome's security architecture, understanding whether the CRLset is effective is of critical importance. Furthermore, Mozilla has announced that Firefox will be moving to a similar architecture in the future [7], which also motivates additional study of CRLsets.

Bibliography

[1] The Directory - Authentication Framework. Internation Telecommunication Union, 2008.

[2] Revocation doesn't worl. ImperialViolet, 2011. https://www.imperialviolet.org/2011/03/18/revocation.html.

[3] CA EV: Revocation checking. Mozilla, 2012. https://wiki.mozilla.org/CA:EV_Revocation_Checking.

[4] How certificate revocation works. Microsoft, 2012. http://technet.microsoft.com/en-us/library/ee619754(WS.10).aspx.

[5] Public WiFi Hotspots Ripe for MITM Attacks. infosecurity, 2013. http://www.infosecurity-magazine.com/news/public-wifi-hotspots-ripe-for-mitm-attacks/.

[6] Android/OpenSSL: Support EV certificates. Google Code, 2014. https://code.google.com/p/chromium/issues/detail?id=117478.

[7] CA:ImprovingRevocation. Mozilla, 2014. https://wiki.mozilla.org/CA:ImprovingRevocation.

[8] Crlsets. The Chromium Projects, 2014. https://dev.chromium.org/Home/chromium-security/crlsets.

[9] Cryptography functions. Microsoft MSDN, 2014. http://msdn.microsoft.com/en-us/library/windows/desktop/aa380252(v=vs.85).aspx.

[10] An evaluaton of the effectiveness of chrome's crlset. Gibson Research Corporation, 2014. https://www.grc.com/revocation/crlsets.htm.

[11] The heartbleed bug, 2014. http://heartbleed.com/.

[12] How certificate revocation (doesn't) work in practice. Netcraft, 2014. http://news.netcraft.com/archives/2013/05/13/how-certificate-revocation-doesnt-work-in-practice.html.

[13] Net market share. Net Applications, 2014. http://www.netmarketshare.com/.

[14] Oid info. Openssl, 2014. http://oid-info.com/get/2.16.840.1.114412.

[15] Openssl - cryptography and ssl/tls toolkit. Openssl, 2014. https://www.openssl.org/.

[16] Openssl documents. Openssl, 2014. https://www.openssl.org/docs/apps/openssl.html.

[17] Ssl crl activity. SANS, 2014. https://isc.sans.edu/crls.html.

[18] Zytrax. Survival guide - TLS/SSL and SSL (X509) certificates, 2014. http://www.zytrax.com/tech/survival/ssl.html.

[19] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1* . IETF, 2006. RFC 4346, .

[20] Z. Durumeric, J. Kasten, F. Li, J. Amann, J. Beekman, M. Payer, N. Weaver, J. A. Halderman, V. Paxson, and M. Bailey. The matter of Heartbleed. In *Proc. of IMC*, November 2014.

[21] C. Ellison and B. Scheiner. Ten Risks of PKI: What you are not being told about Public Key Infrastructure. *Computer Security Journal*, XVI(1), 2000.

[22] S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* . IETF, 2008. RFC 5280, .

[23] A. Malpani, S. Galperin, and C. Adams. *X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP* . IETF, 2013. RFC 6960, .

[24] P. Mutton. Certificate revocation: Why browsers remain unaffected by heartbleed. Netcraft, 2014. http://news.netcraft.com/archives/2014/04/24/certificate-revocation-why-browsers-remain-affected-by-heartbleed.html.

[25] Y. Pettersen. *The Transport Layer Security (TLS): Multiple Certificate Status Request Extension*. IETF, 2013. RFC 6961, .

[26] L. Rehmann. The poodlebleed bug, 2014. http://poodlebleed.com/.

[27] D. Wagner and B. Scheiner. Analysis of SSL 3.0 protocol. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, April 1997.

[28] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In *Proc. of IMC*, November 2009.

[29] L. Zhang, D. Choffnes, T. Dumitras, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proc. of IMC*, Vancouver, Canada, 2014.

[30] L. Zhang and A. Mislove. Building confederated Web-based services with Priv.io. In *Proceedings of the 1st ACM Conference on Online Social Networks (COSN'13)*, Boston, MA, October 2013.