Browser implementation of certificate revocation checks in practice

Viktor Wahlberg
Department of Computer and Information Science
Linköping University
Linköping, Sweden
viktor@wahlberg.dev

***Abstract*—In order to browse the web securely most websites will issue so called SSL certificates, however, how do we know that these certificates have not been revoked by the issuer? Through code analysis, experiments and research the paper shows how the popular browsers Firefox and Chromium handle certificate revocation and explore the benefits and drawbacks of the different approaches. It asserts that none of these systems are perfect today, however, it also refer to other proposed systems that are emerging and may soon be ready for production environments.**

***Index Terms*—Certificate revocation, CRL, OCSP**

I. Introduction

This paper shows how browsers deal with the possibility of a website certificate being revoked. I demonstrate the notable differences between the two browsers Firefox and Chromium on a theoretical level and through experiments. Furthermore, this report highlights the desirable characteristics of good certificate revocation checks and compares them with the currently used systems and future technologies.

A. Purpose

With this paper I seek to answer the following questions:

- How do different web browsers handle certificate revocation checks? (Firefox and Chromium)
- Is certificate revocation always checked and is there a relation between revocation checks and certificate type?
- What are key features of good certificate revocation checks?
- How do CRLSet and OneCRL compare to an actual list of revoked certificates?

B. Background

This paper is conducted as part of the university course *TDDD17: Information Security, second course* at Linköping University. I was tasked with examining how modern browsers handle certificate revocations.

C. Scope and limitations

As this article is written as part of a university course, the time for the assignment is roughly estimated from the number of points awarded upon completion. I expect to spend about 53 hours working on this report, including all the meetings, seminars and the actual writing, researching, analysing and experimenting

The scope is limited to only include Firefox and Chromium. Firefox and Chromium are readily available on all major computer platforms and when accounting for forks it makes up a vast majority of desktop web-browser usage. They are also both open-source and their code is available for download and compilation instructions are available.

This article will focus on desktop users. Mobile devices will not be considered.

II. Theory

A. OCSP

Online Certificate Status Protocol (OCSP) is a standard that allows querying to see if a specific certificate has been revoked. [1]

B. OCSP stapling

OCSP stapling allows for a web server to attach a recent record of a OCSP query, removing the need for every user to perform the OCSP request themselves. This is beneficial for user integrity and reduces the load for OCSP servers. [1]

C. CRL

Certificate Revocation Lists (CRLs) are lists that contain information about a large amount of revoked certificates. [1]

D. OneCRL

OneCRL is a CRL that is maintained and curated by Mozilla and used in their Firefox web browser. Whenever an update is made to the list it is pushed to all Firefox users. [1]

E. CRLSet

CRLSet is a CRL maintained, curated and implemented by Chromium. Whenever an update is made to the list it is pushed to all Chromium users. [2]

F. Firefox revocation checks

Firefox is an open-source web browser and implements OneCRL. If no record is found Firefox instead uses OCSP querying in order to determine its revocation status. These checks are made at least once for every domain you visit during each session. [1]

G. *Chromium revocation checks*

Chromium is an open-source web browser. Chromium implements CRLSet. If no record is found in CRLSet Chromium will assume the certificate is not revoked. [2] If the certificate is a so called *Extended Validation Certificate* Chromium will send an OCSP-request to ensure it has not been revoked.

H. *CRLite*

CRLite, first proposed by J. Larisch et al, is a technology that would allow for a complete CRL to be stored locally for every user without any significant storage and download requirements. By crawling already publicly available and certified certificate revocation sources and creating a cascading bloom filter, it is estimated the total storage requirements would be around 10MB. [3]

As of early 2019 Mozilla is evaluating CRLite as a replacement for OCSP and OneCRL. Currently the Nightly version of Firefox has CRLite implemented, however, it is not yet relied upon by the browser and still uses OneCRL and OCSP. [1] [4]

III. Method

This article will primarily be based on my own analysis of code and experiments that I conduct. It will, however, also be based on already existing scientific literature.

A. *Experiments*

Experiments were conducted partly by using already existing and freely available tools, specifically to gather data. New scripts were also written in order to collect, analyse and sort data.

See Appendix A.

B. *Code Analysis*

Code analysis will be performed on the browsers, following a few simple and repeatable steps:

- Search online to try and find the functions responsible for certificate verification. If that fails, continue.
- Browse the documentation and determine the code structures that are implemented
- Download the code
- Try to identify files of interest, manually finding files with interesting names. Also look inside these files to see what dependencies they have and if any of them are interesting
- Check all interesting functions and hopefully find what was desired

See Appendix B.

IV. Result

A. *Key features*

There are a few key features of certificate revocation checks that are important to most users: [1]

- You want to be certain that the certificate received is not revoked
- Checks must be done fast and should not impact user experience
- User integrity must be preserved

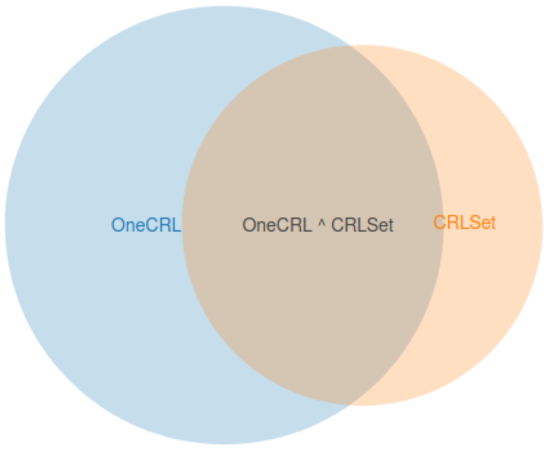

Fig. 1. To scale Venn Diagram of OneCRL and CRLSet revoked certificates

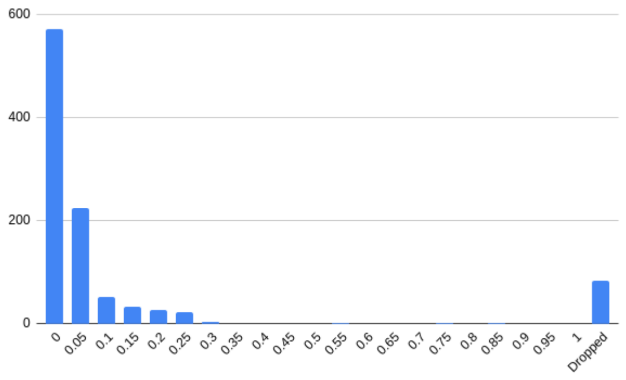

Fig. 2. OCSP-request timings

B. *Experiments*

The three CRLs CRLSet, OneCRL and CRLite were compared and tests were performed in Firefox, Firefox Nightly and Chromium. For extensive reading, see Appendix A.

1) Data comparison: CRLSet contained 928 revoked certificates and OneCRL contained 1373. As is made clear in Figure 1 the two sets are fairly homogeneous, however, they both have many certificate revocations that were not included in the other set. Their intersection were 638 certificates, meaning OneCRL contains 68.75% of CRLSets revoked certificates and CRLSet contains only 46.50%.

2) OCSP timings: Experiments were conducted in Firefox where a number of websites were visited and the OCSP-requests were captured. Firefox is supposed to send an OCSP-request every time OneCRL does not contain the certificate and there is not OCSP-stapling employed by the website.

As can be seen in Figure 2, OCSP-requests were generally completed fairly quickly. We can note from the raw data that all but 4 requests were completed in less than 0.29s. There were also 83 packets that were considered dropped or otherwise containing errors.

3) CRLite: The effectiveness of CRLite was tested and meta-data from the set was gathered. CRLite was able to handle all websites that were visited, always producing a hit in the filter. CRLite contains about 125 million valid and almost 4 million revoked certificates. It will be used as a benchmark for a complete list of valid and revoked certificate, as that is what it strives to be.

C. Code Analysis

The code for two browsers Chromium and Firefox was analyzed to get a better understanding of how they both function. For extensive reading, see Appendix B.

Both browsers acted very similarly when examining the code. Notable things that were looked for were that both browsers had synchronous functions for validation of certificates and revocation checks and no website data was transmitted from either browser before the functions had returned. Notable differences were that Firefox always sent an OCSP-request, unless cached, and Chromium sent OCSP-requests for EV-certificates, unless cached.

V. Evaluation

The experiments that were conducted could have been made using more sophisticated tools. Having a setup where the system could not interfere with the data collection would have been preferable. The list of websites visited could also have been better. The list was acquired by a Google-search and for this report it was considered good enough.

The code analysis portion was handled fairly well, however, there were several shortcuts that had to be made. Firstly, it was not feasible to become familiarized with the entire code base and therefore most of the code went overlooked. Therefore there could have been portions of the code that were of interest that were ultimately not identified. It would probably have been preferred to contact someone who has worked on the project and ask.

VI. Discussion

It is clear to me that OneCRL and CRLSet are both incomplete. I find OneCRL as being entirely unnecessary as it contains so few certificates and in every other case Firefox sends OCSP-requests. The use of CRLSet on the other hand is in no way enough to secure users from potential attacks, yet, it is better than having nothing.

The proposed CRLite, implementing a cascading bloom filter, is to me a good concept. The general characteristics of cascading bloom filters allows for relatively quick and extremely storage-effective are both very good for a CRL implementation. Having the actual certificate meta-data is, as far as I can see, not necessary; a filter is sufficient.

Firefox with its OneCRL and OCSP-request system currently protects its users, and based on the experiments conducted, does it in a timely manner. It does, however, leave open the possibility of integrity violations by OCSP-servers. Since OCSP-requests are sent for every new website, internet usage could be monitored.

CRLite would be a good solution as it enables integrity to be preserved, storage requirements are fairly equivalent to current implementations, checks are relatively fast and you can be confident in the validation process as CRLite contains both valid and revoked certificates.

VII. Conclusion

Chromium and Firefox are very different in handling certificate revocations. Firefox will always check so that the domain you visits have a certificate that has not been revoked. It does this partly by relying on its OneCRL list and pertly by performing OCSP queries. Chromium only ever employ OCSP checks when a domain has an extended validation certificate. Chromium implements a list called CRLSet which is always checked.

In order to employ good certificate revocation checking a browser should care for all the users needs. It needs to be fast, it needs to protect the user and it needs to care for the users integrity. Performing OCSP checks in the way that Firefox does impacts the user integrity by potentially allowing for your traffic to be tracked. Chromium on the other hand could potentially serve websites that use a certificate that is revoked, exposing users to man-in-the-middle attacks.

CRLSet and OneCRL are both very incomplete. They each have about 1000 entries and the total number of revoked certificates are, at the very least, in the tens of millions.

In conclusion there is still much work to be done in the field of browser revocation checks. No major browser implements a solution that is entirely satisfactory, even though there are proposed new technologies that may soon change this.

References

[1] *"CA/Revocation Checking in Firefox - MozillaWiki"*. Accessed on: Apr. 18, 2021. [Online]. Available: https://wiki.mozilla.org/CA/Revocation_Checking_in_Firefox

[2] *"CRLSets - The Chromium Projects"*. Accessed on: Apr. 18, 2021. [Online]. Available: https://dev.chromium.org/Home/chromium-security/crlsets

[3] 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,"* 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2017, pp. 539-556, doi: 10.1109/SP.2017.17.

[4] *"The End-to-End Design of CRLite - Mozilla Security Blog" "*. Accessed on: Apr. 19, 2021. [Online]. Available: https://blog.mozilla.org/security/2020/01/09/crlite-part-2-end-to-end-design/

[5] https://searchfox.org/mozilla-central/source/browser

Appendix A
Experiments and data analysis

This document will contain data relating to CRLSet, OneCRL and CRLite as well as results from experiments conducted regarding CRLs and relating technologies.

Data collection will be conducted by utilizing already existing and publicly available tools. Further analysis will be made using specialized scripts written specifically for the analysis. Under each subtitle for the result will be first the means of data-collection, followed by the data, followed by notable attributes of the data collected.

TABLE I
NUMBER OF CERTIFICATES CONTAINED IN EACH LIST

	CRLSet	OneCRL	CRLite
Revoked	928	1,372	3,911,734
Valid	-	-	125,075,740
Disk Size	<1MB	<1MB	<6MB

TABLE II
COMPARISON OF CRLSET AND ONECRL CERTIFICATE REVOCATIONS

	CRLSet	OneCRL
Total Revoked	928	1,372
Overlapping	638	638
Unique	290	678

A. Metadata

In the following data collection, each of the 3 CRLs were compared based on disk size, the number of contained known revoked certificates and the number of contained confirmed not revoked certificates. The disk size was measured in whole megabytes (MB). Notably, only CRLite allows for checking confirmed not revoked.

Data collected on the 13th of April 2021 and data is found in Table I.

B. Data-comparison of OneCRL and CRLSet

In the following analysis I looked at how many overlapping certificates were found in each of the currently implemented CRLs. This was done by first extracting the data from each CRL and then comparing them using a script.

The comparison was made on serial numbers. This makes for the possibility of false positives since there could be duplicate serial numbers. This was made in the interest of time.

Data was collected on the 13th of April 2021 and data is found in Table II.

C. OCSP request timings

Firefox send OCSP-requests for all certificates that are not contained in OneCRL or has OCSP-stapling. 500 websites were visited and traffic analysed in Wireshark with the OCSP filter turned on. The websites were gathered from http://moz.com/top500. Of the 500 websites, with the OCSP-filter enabled, there were 937 HTTP OCSP-requests needed. There were a total of 658 OCSP staplings. The average time for the HTTP OCSP-requests were 0.041s. The average extra time per website required for OCSP-requests were 0.077s, given the response was not dropped. The longest OCSP-request took 0.828s, however, all but 4 requests were completed in less than 0.29s. 83 packages were marked by Wireshark with ”TCP Previous segment not captured” or ”TCP Spurious Retransmission”, these were assumed to be timed out or otherwise dropped packages.

D. OCSP requests needed with CRLite

Accessing the same 500 websites with Firefox Nightly found that all certificates were available from CRLite and no OCSP-request would have been necessary. Firefox Nightly still sends OCSP-requests, however, and I can access the hits and misses to CRLite from its Telemetry feature.

APPENDIX B
CODE ANALYSIS

A. Methodology

I first browsed the official websites related to each browser in order to find out what general structural guidelines were implemented for the code. I specifically looked for anything that would be related to security, certificates or connectivity. From there I downloaded the code and looked for files in the appropriate folders with names suggesting it would be relevant to the analysis.

In order to get a better understanding of how the code operated, and to make sure that it was indeed doing what I thought it was, I compiled the project and began testing. Console output and time delays were implemented to determine what code ran when and if it was running on the same thread.

B. Firefox

I noted a file called /security/certverifier/CertVerifier.cpp as probably being responsible for general checking of certificates. In order to verify this I added a delay and console print to the execution of the function VerifyCert. Building and running the browser revealed a significant delay for every website loaded.

In testing I found that all none-cached requests were caught in the VerifyCert function and were executed synchronously. It was also noted that no actual website data was sent before the function was returned. This was checked using Wireshark and loading a website that were under my control.

C. Chromium

I noted a file called /net/cert/internal/cert_verify_proc.cc as probably being responsible for general checking of certificates. It notably includes OCSP checks and CRLSet checks. In the beginning of the related h-file, I can read that the purpose is to “perform certificate path building and verification for various certificate use”. In order to verify its use I added a time delay with console output to the *Verify* function, compiled the code and ran the browser. I noted a significant delay before the page loaded and a console output marking that our time delay was running. I also noted, by using Wireshark, that the transmission of website data seems to be delayed until the *Verify* function is completed.

In testing I found all new requests I sent were caught in the *Verify* function and were always synchronous. The only time I noted it was not executed was when a previously visited website with an extended validation certificate was accessed, presumably because it is now cached.

Make note that I have confirmed that Chromium indeed performs an OCSP check to validate EV certificates. This is different from what I were able to read online on the Chromium website.

I added a single console output at the start of the *Verify* function and one at the very end. I visited sites I had control over that only contained a single domain and accompanying certificate. The typical time between these console outputs was 30-50ms. This was similar, although notably always slightly more or less, to the SSL loading time I found when checking the browser console's networking tab. This result was captured on a debuggable version of Chromium, which is notably slower than a production version. Running the official distributed binary of Chromium and checking SSL times in the networking tab showed a significantly faster response, typically 15-20ms.