Masaryk University
Faculty of Informatics

Network Traffic Analysis of OCSP Protocol

Bachelor's Thesis

Ján Štefkovič

Brno, Spring 2017

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ján Štefkovič

Advisor: doc. Ing. Pavel Čeleda, Ph.D.

Acknowledgement

I would like to thank doc. Ing. Pavel Čeleda, Ph.D. for leading the general course of this work and help with academic typesetting. I also thank Mgr. Martin Laštovička for many consultations, regular advice and answering questions.

Abstract

This bachelor thesis provides analysis of OCSP protocol, which is used for revocation status checking of X.509 certificates. It examines the structure of OCSP requests and responses, describes the behavior of common OCSP clients and explains how to monitor OCSP traffic on a computer network. It also describes some of OCSP issues, such as privacy and security. Finally, it demonstrates some of these problems by showing examples.

Keywords

OCSP, X.509, certificate revocation, network traffic analysis

Contents

List of Tables

List of Figures

1 Introduction

Computers are a key part of our everyday life, mainly in the working sphere. Times of standalone machines lasted for only a short amount of time. Interconnecting computers was necessary to allow communication and sharing of information. However, connecting devices with the outer world also makes them vulnerable to attacks.

Some types of attacks, such as impersonation attack, are carried out by means of imitating another entity. For an entity to prevent this and to prove its own identity to a client, digital certificates of Public Key Infrastructure (PKI) are used. These certificates sometimes need to be revoked (immediately invalidated) and client needs a method to check whether a certificate has been revoked. Online Certificate Status Protocol (OCSP) is one of the ways to do that.

The goal of this work is a network traffic analysis of OCSP protocol. The background part of the thesis explains digital certificates, their issuing and the possibility of revocation. It shortly talks about the older revocation status checking method called Certificate revocation lists (CRLs). Then follows the core of this work, an OCSP protocol analysis.

The operation of OCSP, structure of its requests and responses and various implementations of this protocol are carefully examined. This work then shortly explains network monitoring, which is necessary for our study of OCSP use in practice. The method to capture OCSP traffic is developed with the help of the theoretical knowledge of OCSP and then proven in practical experiments.

OCSP has several issues, some of them solved by later improvements to this protocol. Particular issues were chosen and demonstrated in practical experiments and examples.

2 Certificate revocation

This chapter describes the purpose and operation of certificate revocation. At first, it explains what is a certificate of Public Key Infrastructure. Then it introduces an older revocation status checking method called Certificate revocation lists.

2.1 Digital certificates of Public Key Infrastructure

Digital certificates of Public Key Infrastructure are means to ensure confidentiality and integrity between entities.

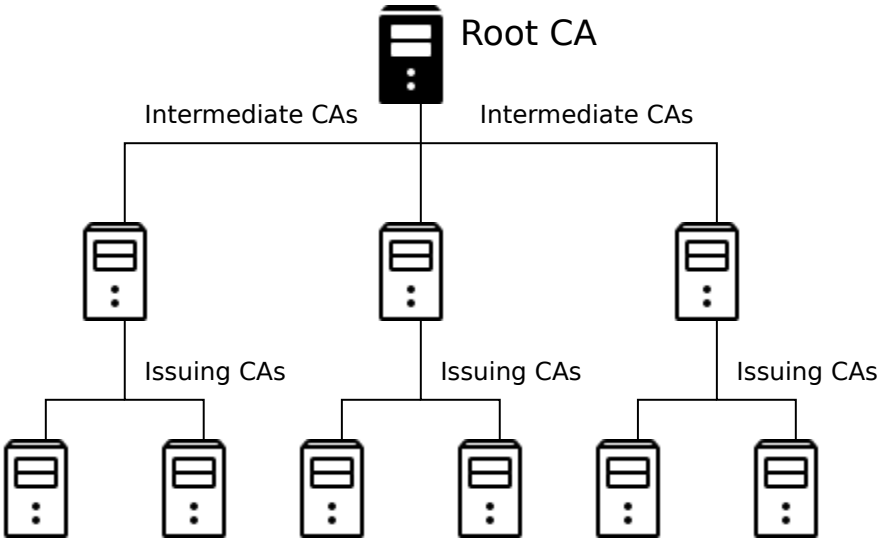

Figure 2.1: Certificate Authority hierarchy

In PKI, a trusted third party called a registration authority verifies the identity of a person or entity. Then it instructs the certificate authority (CA) to issue a digital certificate which contains that entities' public key. CA declares this certificate valid by signing it with its own root certificate. This certificate (and the public key contained therein) may subsequently be used to prove identity and enable secure communication with other parties [1].

CAs are organized in a hierarchical structure, as shown in Figure 2.1. On the top is the Root CA, whose certificate is a self-signed certificate. Every other CA has its certificate signed by the higher-level CA. A list of root certificates is present on every system that will use the PKI, so the system can verify if the certificate is signed by a trusted CA. If not, the certificate chain is followed up to the higher CA, until a trusted CA is found. This process ends at Root CA [2].

The structure of the digital certificates is in conformance with the X.509 standard [3]. The required fields creating the certificate are as follows [1]:

- **Issuing CA** – the name of the certificate authority which issued the certificate.
- **CA Digital Signature** – the digital signature of the issuing certificate authority.
- **Version Number** – specifies the version of X.509 to which the certificate conforms (current version is 3).
- **Serial Number** – a unique number identifying the specific certificate issued by a particular CA.
- **Subject/Owner** – the owner of the certificate (a person, company, application etc.).
- **Owner's Public Key** – the public key associated with the certificate and corresponding to the certificate owner's private key.
- **Validity Period** – the dates during which the certificate is deemed to be valid.
- **Certificate Usage** – specifies the approved uses of the certificate.
- **Signature Algorithm** – the hashing and digital signature algorithms used in the creation of the certificate.

Digital certificate expires after its validity period. However, it is often desirable to make a certificate invalid, for example, when the private key was compromised. This is called revocation. Since users would deem a certificate valid till it expires, there must be a mechanism to check whether the certificate has not been revoked.

2.2 Certificate revocation lists

Certificate revocation list (CRL) is a time-stamped list of serial numbers of certificates that have been revoked. Entities presenting revoked certificates should no longer be trusted. RFC 5280 [3] defines two states of revocation:

- **revoked** – irreversible revocation (in the case of the private key being compromised, an improperly issued certificate, change of name, etc.),
- **hold** – reversible status noting temporary invalidity of the certificate.

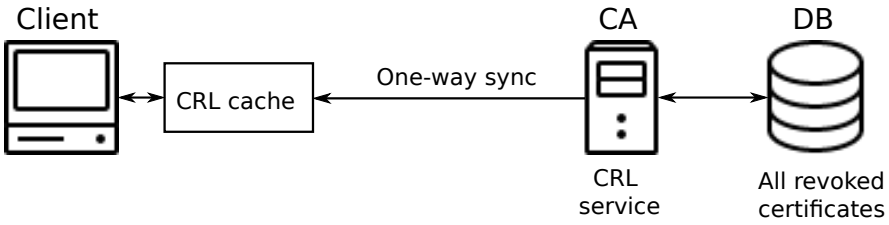

Figure 2.2: CRL verification operation

A CRL is a signed data structure periodically published by the CA which issued the corresponding certificates. It can also be published immediately after revocation of some certificate. CRLs are valid during a lifetime, e.g., one hour, one day, or one week. It is made freely available in a public repository. A client-side application performs the verification, as depicted on Figure 2.2. The distribution point from where to fetch the CRL is in a certificate's extension. The time of the next update is contained in CRL.

Drawbacks of this approach are that CRLs can grow huge in size over time. Also, CRL can be unavailable at the time needed or a certificate can be revoked between client's periodical sync.

Another method for certificate revocation is OCSP. As a core of this work, it is described in the next chapter. An important concept of OCSP Stapling is explained in Section 5.2 after noting problems of standard OCSP.

3 OCSP protocol

This chapter describes Online Certificate Status Protocol in detail. It examines the structure and the main attributes of its requests and responses. For detailed ASN.1 specifications see RFC 6960 [4]. However, it is only a specification document and actual implementations may differ.

3.1 OCSP operation

Online Certificate Status Protocol works as a request-response system, as visible on Figure 3.1. Revocation status of a certificate is requested on demand, providing the client with the most actual information. An OCSP client issues a status request to an OCSP responder (server) listed in the certificate. Name of this responder is contained in the Authority Information Access extension, in the *accessLocation* field. Acceptance of the questioned certificate is suspended till a response is received. There is also no need to store any lists on client's side as by CRLs, although responses can be cached thanks to inserted time information. The protocol can be implemented over various transportation or application layers. Preferred is HTTP.

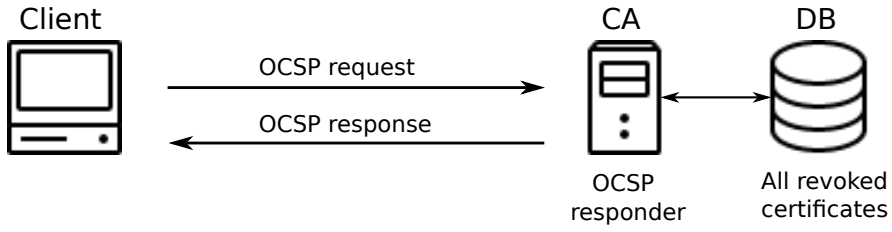

Figure 3.1: OCSP request-response operation

The certificate status value in a response is one of the three following:

- **good** – no certificate with the requested certificate serial number currently within its validity interval is revoked,

- **revoked** – the certificate has been revoked, either temporarily (the revocation reason is certificateHold) or permanently,
- **unknown** – the responder doesn't know about the certificate being requested, usually because the request indicates an unrecognized issuer that is not served by this responder.

Certificates with *revoked* status should be rejected, *unknown* status leaves the possibility to try another source of information. OCSP responder can also return an error message if a problem occurs.

3.2 Request and response structure

Request A request consists of a protocol version, service request, target certificate identifier and optional extensions. OCSP responder checks if the message is well formed, contains all information needed and if the responder is configured to provide the requested service. If some condition is not met, the responder produces an error message, otherwise, it returns a response.

HTTP-based OCSP requests can use either the GET or the POST method to submit their requests. To enable HTTP caching, small requests (that after encoding are less than 255 bytes) may be submitted using GET. If HTTP caching is not important or if the request is greater than 255 bytes, the request should be submitted using POST [4].

Response A response consists of version of the response syntax, identifier of the responder, time when the response was generated, responses for each of the certificates in a request (consisting of target certificate identifier, certificate status value, response validity interval and optional extensions), optional extensions, the signature algorithm used and the signature of a hash of the whole response. Responses can contain the following four times. They can be used for caching:

- **thisUpdate** – the most recent time at which the status being indicated is known by the responder to have been correct,
- **nextUpdate** – the time at or before which newer information will be available about the status of the certificate,

- **producedAt** – the time at which the OCSP responder signed this response,

- **revocationTime** – the time at which the certificate was revoked or placed on hold.

3.3 Lightweight OCSP profile

OCSP responders need to work in real time and deal with high volume of requests. Normal deployments operate well on powerful systems without constraints on bandwidth and processing power. The problem is with mobile environments, where minimal bandwidth usage and client-side processing is desired. Relying parties (users) need to ensure the status of a certificate.

The lightweight OCSP profile is specified in RFC 5019 [5]. It is meant to address the scalability issues inherent when using OCSP in large scale (high-volume) PKI environments. It also can be used when the lightweight solution is required to minimize communication bandwidth and client-side processing. RFC 5019 defines a message profile and clarifies OCSP client and responder behavior that will permit:

- OCSP response pre-production and distribution (pre-produced signed responses specifying the status of certificates at a specified time),

- reduced OCSP message size to lower bandwidth usage,

- response message caching both in the network and on the client.

Attributes of OCSP request conforming to this standard are:

- OCSP request includes only one request in the OCSPRequest.RequestList structure,

- SHA1 must be used as the hashing algorithm for the CertID.issuerNameHash and the CertID.issuerKeyHash values,

- the client should attempt to contact the OCSP responder first, CRL is attempted only after configured timeout and number of retries (to contact OCSP responder),
- requests that are less than or equal to 255 bytes in total (after encoding), including the scheme and delimiters (http://), server name and base64-encoded OCSPRequest structure are sent by the GET method (to enable OCSP response caching), OCSP requests larger than 255 bytes should be sent by the POST method.

Clients have to determine responders conformant with this profile with the help of out-of-band mechanisms. However, interoperability will occur between a fully conformant OCSP 6960 client and OCSP 5019 responder [5].

3.4 Implementation

OCSP is implemented in various operating systems and applications. For example, ocsp is a utility in OpenSSL (Cryptography and SSL/TLS Toolkit [6]). Client-side support for the OCSP has been added in the release of Public Key Infrastructure Enhancement for version 5.0 of the Java 2 platform [7]. Security.framework of OS X uses ocspd – OCSP and CRL Daemon to cache and network fetch CRLs and OCSP responses [8].

This section describes the implementation of OCSP client in MS Windows as an example from the proprietary area and by Mozilla Network Security Services for the open-source area.

Microsoft CryptoAPI

Cryptography API: Next Generation (CNG) or in short CryptoAPI is application programming interface that enables application developers to add authentication, encoding, and encryption to MS Windows-based applications [9]. It is used by developers of applications that will enable users to create and exchange documents and other data in a secure way, especially over nonsecure media such as the Internet [10].

The OCSP client in Windows is a component that generates OCSP requests based on information stored in the Authority Information

Access (AIA) extension of the certificate it is validating [11]. AIA extension allows SSL/TLS clients (mostly web browsers) to go get the missing intermediate certificates, not presented by the server. This extension, that adds in the final certificate a "CA Issuer" containing a URL, allows the browser to find the missing certificate and to check the chain again [12].

The Windows OCSP client supports the Lightweight OCSP Profile. According to MS documentation [13], OCSP is usually preferred over CRL.

Mozilla Network Security Services

Network Security Services (NSS) by Mozilla is a set of libraries designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards [14].

NSS uses a global setting to have OCSP either enabled or disabled. If code execution attempts to use OCSP to verify a certificate, NSS will:

- open a connection to the OCSP server using HTTP,
- send a request, receive a response,
- verify the validity (signature, freshness) of a response.

Only after all of the above succeeds then NSS will make use of the received response [15]. As of today NSS always uses HTTP POST when talking to OCSP servers. Using POST prevents caching of OCSP responses in proxy servers. It has been proposed that NSS shall support HTTP GET. An application shall be able to instruct NSS to use GET by default.

Others

Later Section 5.1 deals with privacy issues of OCSP, regarding mostly the revealing of User-Agents. Therefore is here stated a few more facts about OCSP implementation in common applications, that were determined in previous studies [16].

Firefox uses its own User-Agent because it uses its own encryption library (NSS). MS IE and Safari use the functionality of their respective operating systems for OCSP lookups, not directly revealing their user-agents.

Chrome uses OCSP only for Extended Validation (EV) certificates, again using operating system, except Linux, where it uses its own User-Agent. EV is such a certificate that requires verification of the requesting entity's identity by a CA [17]. iOS also uses OCSP for EV certificates only [18].

The study by Liang Zhu et al. [16] states, that not all clients followed the standard for creating OCSP GET request (they skipped the URL-encoding). These requests were created mostly by yet already fixed bug and were accepted by servers anyway.

4 OCSP traffic analysis

This work studies OCSP traffic and related issues. To do this, we must be able to detect it. In this chapter, we examine the proposed structure of OCSP requests and responses and their implementation. We use this knowledge to compose traffic filters. This chapter describes HTTP-based OCSP requests/responses composed of HTTP header and OCSP body (payload), as depicted in Figure 4.1.

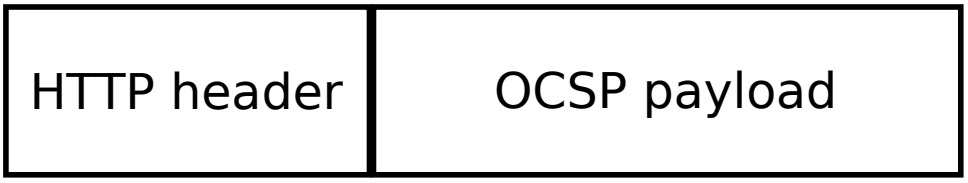

Figure 4.1: HTTP-based OCSP request/response

4.1 Network monitoring

Network traffic measurement and analysis is the process of reviewing, analyzing and managing network traffic for any abnormality or process that can affect network performance, availability and/or security. In this work, we monitor the traffic of OCSP. To do so we use packet analysis.

Network monitoring can be done in active or passive ways. In this work, we use passive approaches. Passive approaches observe only the actual traffic at some measurement point. They do not inject any own traffic or modify the actual traffic. Passive monitoring can be done with the help of a packet capture program. It deals with information such as the accurate bit or packet rates, timing, protocol traffic [19].

There are two ways to gather information on networks – Host-based monitoring and Network-based monitoring. Host-based monitoring is ensured via the agent that operates on a particular host. It does not require extra hardware and can see most of the host's activities. The second way is to gather information from the infrastructure. Network-based monitoring is a process of monitoring all incoming

and outgoing data from and to the network. This is done by placing network monitoring probes.

Methods then differ according to what is being collected. Deep Packet Inspection (DPI) examines the packets one by one. It examines the data part (and possibly the header) of a packet as it passes the inspection point. Examining packets one by one to the depth is effective for certain tasks only, e.g. finding intrusions. Since analysis of packet headers can be done economically (locations of packet header fields are restricted by protocol standards) [20], flow monitoring can be chosen, the last and the most preferred version being IPFIX [21] (Internet Protocol Flow Information eXport). Flow monitoring also examines all packets but uses just their headers to aggregate them into flows. Flow is a set of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties (RFC 7011 [21]).

However, this work needs to use packet analysis, since we inspect information of both packet headers and bodies. We use the following tools:

Wireshark is probably best-known network protocol analyzer, used both in industrial and educational environments. It provides an in-depth picture of examined networks, allows various filters and has graphical user interface [22]. We have used it for the first analysis or anytime we needed to inspect a packet.

tshark is a terminal-based version of Wireshark. It lets you capture packet data from a live network, or read packets from a previously saved capture file and print a decoded form of those packets. *tshark's* native capture file format is *libpcap* format with proposed file extension *.pcap*, which is also the format used by *tcpdump* and various other tools. It is a program run from command-line. For example of *tshark* usage see Appendix A.2. In this work, we use it for continuous captures of traffic.

tshark network monitoring tool allows one to capture desired communication and do general observation. For deep inspection of a packet with the help of GUI, the optimal tool is Wireshark.

In this work, we demonstrate OCSP privacy issues by using it to identify devices. Identification of devices and operating systems on a network can be done in either active or passive way. The principle of the passive method is just to listen at some point in the network. This

point is usually a place which aggregates whole communication. We can also deploy more listening devices on various points.

Example passive method used in this work is to capture HTTP User-Agent on ISO/OSI layer 7. The User-Agent header is part of HTTP protocol that tells a website information about the application (such as a browser) and operating system. This is done so the website can provide customized content [23]. This work could use it because OCSP requests are carried over HTTP.

4.2 OCSP request

HTTP-based OCSP requests use either GET or POST method. Both must be detected in a different way.

POST An OCSP request using the POST method is constructed as follows: the Content-Type header has the value "application/ocsp-request", while the body of the message is the binary value of the DER encoding of the OCSPRequest. That means, in the case of POST request, we can filter simply by http.content_type.

An example of captured POST request header looks like following (it is followed by the body - encoded OCSP request):

```
POST /ocsp HTTP/1.1
Host: clients1.google.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
   :44.0) Gecko/20100101 Firefox/44.0
Accept: text/html,application/xhtml+xml,application/xml;q
   =0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Length: 75
Content-Type: application/ocsp-request
Connection: keep-alive
```

Listing 4.1: OCSP POST request

GET An OCSP request using the GET method is constructed as follows:

```
GET {url}/{url-encoding of base-64 encoding of the DER
   encoding of the OCSPRequest}
```

Listing 4.2: OCSP GET request format

where {url} OCSP client derives from the value of the authority information access extension in the certificate being checked for revocation, or other local configuration of the OCSP client. So in the case of GET, OCSP request is DER encoded, then base-64 encoded, then url-encoded and this is preceded by GET {url}/. An example of captured GET request looks like following:

```
GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBQ%2FYHKj6JjF6UBieQioT
   YpFsuEriQQUtnf6aUhHn1MS1cLqBzJ2B9GXBxkCEAm3Sf1%2
   FC0kWygVWVs%2F22YI%3D HTTP/1.1
Connection: Keep-Alive
Accept: */*
User-Agent: Microsoft-CryptoAPI/10.0
Host: ocsp.ws.symantec.com
```

Listing 4.3: OCSP GET request

If some unique attribute of OCSP request is carried over to final string, we should be able to construct a filter for this attribute and then capture OCSP GET requests.

Sample OCSP request body looks like the following:

```
SEQUENCE {
  SEQUENCE {
    SEQUENCE {
      SEQUENCE {
        SEQUENCE {
          SEQUENCE {
             OBJECTIDENTIFIER 1.3.14.3.2.26 (id_sha1)
             NULL
          }
          OCTETSTRING
             f2e06af9858a1d8d709b4919237aa9b51a287e64
          OCTETSTRING 4
             add06161bbcf668b576f581b6bb621aba5a812f
```

```
          INTEGER 0x30c9a33fb239662b
        }
      }
    }
  }
}
```

Listing 4.4: OCSP request body

Its HEX form:

```
30 49 30 47 30 45 30 43 30 41 30 09 06 05 2b 0e 03 02 1a 05
00 04 14 f2 e0 6a f9 85 8a 1d 8d 70 9b 49 19 23 7a a9 b5 1a
28 7e 64 04 14 4a dd 06 16 1b bc f6 68 b5 76 f5 81 b6 bb 62
1a ba 5a 81 2f 02 08 30 c9 a3 3f b2 39 66 2b
```

- "30" in the first part represent ASN.1 Constructed type SEQUENCE. Numbers between them are the length of the sequence in bytes [24],
- 06 05 2b 0e 03 02 1a 05 00 represents HashAlgorithm, with 2b 0e 03 02 1a being Algorithm Id,
- 04 stands for OCTETSTRING and 02 stands for INTEGER,
- values in between represent unchanged values as seen in sample request.

Now such an information must be found, that is unique for every OCSP request in HEX form and stays unique after series of encodings.

1. SEQUENCE is encoded as 30 (in hex)
2. first three bytes (in base 16) are: |30|??|30|
3. in binary: |00110000|????????|00110000|
4. Base64 encoding converts it to characters
5. at first align bits to 6-tuples: |001100|00????|????00|110000|
6. then assign character to each 6-tuple: |M|??|??|w|

This means that every OCSP request has the following form:
`GET {url}/M..w{more characters}` and GET requests can be filtered by urls, that contain **/M..w** as a substring (dot represents any character). Simply, all OCSP requests can be captured by setting the filter as:

```
http contains "application/ocsp-request" or http.request.
   uri matches "(.)*/M..w(.)*"
```

4.3 OCSP response

An HTTP-based OCSP response is composed of the appropriate HTTP headers, followed by the binary value of the DER encoding of the OCSPResponse. The Content-Type header has the value "application/ocsp-response". The Content-Length header should specify the length of the response. Other HTTP headers may be present and may be ignored if not understood by the requestor. We can filter OCSP responses by http.content_type.

An example of captured response header looks like following (it is followed by the body – encoded OCSP response):

```
HTTP/1.1 200 OK
Server: nginx/1.10.2
Content-Type: application/ocsp-response
Content-Length: 1736
content-transfer-encoding: binary
Cache-Control: max-age=421159, public, no-transform, must
   -revalidate
Last-Modified: Tue, 3 Jan 2017 06:31:22 GMT
Expires: Tue, 10 Jan 2017 06:31:22 GMT
Date: Thu, 05 Jan 2017 09:35:55 GMT
Connection: keep-alive
```

Listing 4.5: OCSP response example

5 Issues of OCSP and OCSP stapling

The first part of this chapter describes problems of OCSP, using HTTP browsing as an example. During normal operation of OCSP, when a browser is presented with a certificate, it requests the issuing CA for the revocation status of this certificate. CA responds with a signed assertion stating whether the certificate is still valid or if it has been revoked [25]. This operation has several drawbacks. Some of them are solved by OCSP Stapling, which is described in the second part of this chapter.

5.1 Problems

Privacy implications

OCSP reveals information about which certificate is used within the PKI also with the time of use. The client can be identified by IP address and in the case of the client required to sign a request, also by the certificate itself. All this information is leaked to the CA. CA then knows what site the user is visiting and what is user's IP address. This information is also visible to anybody who can capture the traffic.

Man in the Middle (MitM) is a type of an attack when there is a device relaying communication between two parties. MitM can eavesdrop or alter this traffic. In the case of OCSP, an attacker can also learn behavior and identity based on these requests.

The signed requests can be used in replay attacks, too [26]. A replay attack is achieved by using the replay of messages from the original context into the same or the different one. In the case of OCSP, an attacker can mimic one's behavior, trying to impersonate him.

And as is demonstrated in Section 6.1 of this work, OCSP traffic can be used for device and browser identification, even with HTTPS-only functionality turned on. This functionality is achieved for example with the help of HTTPS-only browser addon (such as HTTPS Everywhere [27]). User-Agent is encrypted in HTTPS communication, but OCSP requests are carried over HTTP, thus having visible User-Agent. OCSP provides an attacker with User-Agents captured from OCSP requests. For example, at least one OCSP request is sent every time

a secure web page is accessed. E-mail client such as Thunderbird also produces its own OCSP requests.

Connection slow-down

OCSP traffic slows down the connection. Everytime browser encounters a new certificate, it has to send a request to CA server. CA provides responses to each client that requests certificate status information in real time. This is demanding when a high traffic website is in concern.

The median time for a successful OCSP check was around 300 ms and the mean was nearly a second (in 2012). This delays page loading and discourage sites from using HTTPS [28].

OCSP checks timed out about 15 % of the time and took about 350 ms (in 2015) when succeeding, as states M. Goodwin (Mozilla) [29].

According to the study by Liang Zhu et al. [16], median latency of OCSP queries was 291 ms in 2012, with improvement to 20 ms in 2016.

Over 30 % of the TLS overhead comes from CRLs and OCSP operation. Also, the content delivery cannot take place until the revocation check is done [30].

DoS and its consequences, soft-fail

Denial of Service (DoS) is possible also in relation to OCSP traffic [31].

In the case of CA server unavailability, a browser has to choose between terminating the connection (secure behavior, but decreasing usability) or continuing the connection (risky behavior interfering with confidentiality, ignoring the point of revocation checking). Continuing the connection is referred to as soft-fail. For example, Firefox currently continues the connection by default. The about:config option security.OCSP.require can be set to true to have Firefox terminate the connection instead (verified on Firefox 52.0.1).

This type of attack can be successfully used when it is too hard for an attacker to disable his target with DoS, for example when the target's network is too large. Disabling CA servers can be easier, causing the clients to stop trusting the target's servers.

SHA-1 usage

SHA-1 is a hash algorithm used in the Lightweight OCSP Profile [5], which is implemented in the MS Windows OCSP client. Others use it too, e.g., we detected SHA-1 being used in requests generated on Ubuntu 16.04, Firefox 52.0.1. The study by Liang Zhu et al. [16] states, that although clients can choose the hash algorithm for OCSP requests, all captured requests during the study showed a choice of SHA1.

However, SHA-1 is considered outdated and has been proven to be insecure by generating a collision [32]. A collision – two different documents hashing to the same digest, can be crafted by a well-funded attacker. He can then deceive systems that rely on hashes into accepting a malicious file. SHA-1 should be replaced by safer cryptographic hashes such as SHA-256 and SHA-3. Currently, we don't know about any possible attack taking advantage of this vulnerability in OCSP.

Other problems

In the case of a CA private key compromise, the OSCP may return "revoked" for all certificates falling under that CA [26]. Another problem is a captive portal (e.g. hotel WiFi network) that frequently requires a user to sign in on an HTTPS site but blocks traffic to all other sites, including the CA's OCSP servers [28].

5.2 OCSP Stapling

Stapling concept

OCSP Stapling – or formally, TLS Certificate Status Request extension – is a solution or mitigation to some problems of OCSP.

OCSP Stapling operates differently than standard OCSP. The user usually doesn't ask CA about the particular certificate. Instead, the site itself ask the CA periodically for a signed assertion of certificate status and includes this statement in the handshake at the forming of new HTTPS connection with the client, as visible on Figure 5.1. In other words, we let the web service request the status of a certificate in regular intervals and staple this digitally time-stamped signed result to the handshake with the client [26]. User's browser only verifies this

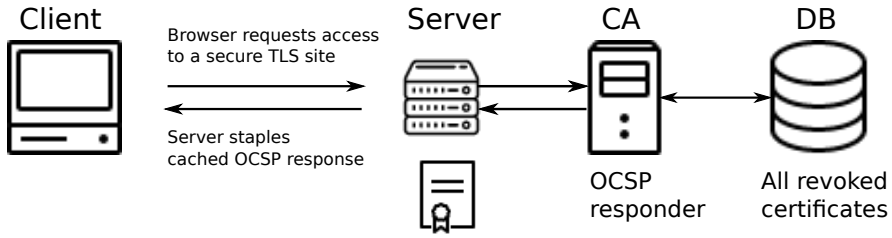

Figure 5.1: OCSP Stapling

signed, stapled response. If not satisfied, it terminates the connection. If there is no stapled response, the client does the revocation check by itself [25].

Solved drawbacks of standard OCSP operation

Here is a list of some issues which are (partially) solved by OCSP Stapling:

- MitM would not hold up because only web service is communicating with CA server,
- for the same reasons, the privacy issue is resolved, the time interval information prevents a replay attack,
- infrastructure costs are cut down because OCSP response can be reused,
- although revocation status is not absolutely fresh, it's still usually fresh enough, and moreover, OCSP-stapling substitutes unavailable OCSP responder,
- captive portals can check their own certificates, so there is no need to visit currently blocked OCSP servers.

OCSP Stapling is supported by common browsers and on servers like Microsoft Server, Apache HTTP Server, and Nginx [33] [34].

There are two issues with OCSP Stapling. First, there is no functionality to request the status information about intermediate CA certificates. Second, the current format of the extension and requirements

in the TLS protocol prevent a client from offering the server multiple status methods. Both of these issues are addressed with Multiple Certificate Status Request Extension [35].

Must-Staple concept

It was already stated that most browsers continue in connection when the revocation check was not successful. This is called soft-fail and is meant to mitigate decreased usability. However, it is clearly not secure and makes revocation checking pointless.

Users can opt out this default behavior and turn on OCSP hard-fail (for example in Firefox). Hard-fail means that the connection is terminated (while soft-fail just tells the user that something is wrong, but continues anyway). In other words, when the revocation status of a certificate is not known, soft-fail means accepting the certificate, while hard-fail means rejecting it.

OCSP Must-Staple makes use of the recently specified X.509v3 Transport Layer Security (TLS) Feature Extension [36]. When a CA adds this extension to a certificate, the browser is required to ensure a stapled OCSP response is present in the TLS handshake. If it is not present, the connection will fail (can result in the non-overridable error page, for example, Firefox since v45, 2016 [29]). The purpose of this extension is to prevent downgrade attacks. Such attacks force to abandon a more secure high-quality mode of operation in favor of a less secure mode of operation, that exists for backward compatibility. The idea is to inform clients that an OCSP response is always stapled. This results in hard-fail when the response is not stapled, preventing otherwise possible DoS attack [36].

There is another solution to certificate revocation, based on short-lived certificates. If a certificate has a lifetime shorter than preconfigured value, it is assumed valid and it is not checked [29].

6 Demonstration of issues detection

We have chosen some of the problems and run our own experiments. In this chapter, we provide step-by-step instructions and our own results. We also explain when and why our experiments went wrong.

6.1 Detection of privacy issues

This section demonstrates identifying a device by capturing and examining outgoing OCSP requests. Certificate status verification is done often because it is a key part of secure communication. OCSP is also preferred before CRLs and OCSP requests are transported mainly by HTTP, as explained in Section 5.1.

The following subsections describe our analysis of OCSP traffic, starting from simple Wireshark capture and observations, continuing with monitoring our own traffic with the help of tshark and finally, high-volume laboratory traffic analysis and statistics.

Wireshark packet capture

Wireshark is a widely used network protocol and traffic analyzer, allowing also Deep Packet Inspection [22]. Wireshark v2.0.2 allowed us to capture traffic, write it to a pcap file and later analyze. We could also read pcap files gathered in another way.

In this work, we have utilized virtual machines. We have used VirtualBox v5.0.24 [37] to run images of various versions of OSs (MS Windows 7, 8.1, 10 and Ubuntu 14.0.4), so we would have controlled environments. VirtualBox provides built-in monitoring, which can be turned on with (modification of) the following command:

```
# VBoxManage modifyvm "ubuntu" --nictrace1 on --
   nictracefile1 file.pcap
```

Listing 6.1: Turn on VirtualBox built-in monitoring

We have monitored the system since the start. Then we have accessed three web pages (from bank sector) over HTTPS and closed the browser. This sample was taken twice for each OS. A scheme of testbed can be seen in Figure 6.1.

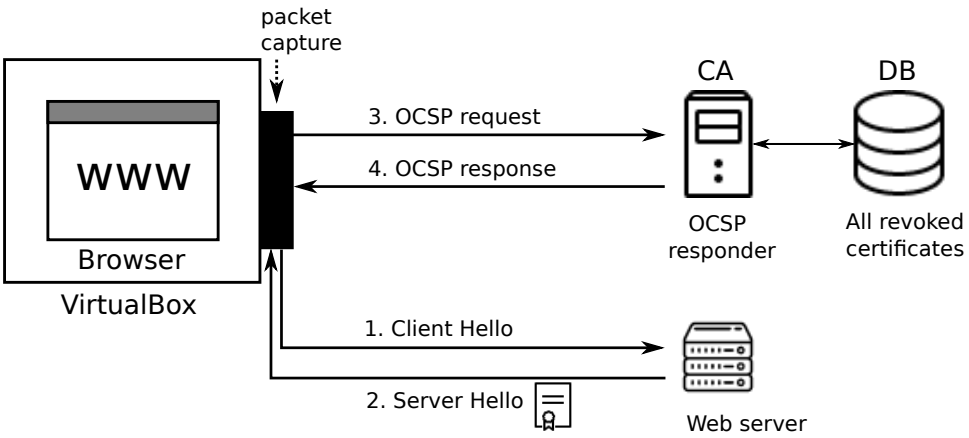

Figure 6.1: Testbed for capturing of OCSP traffic

Then we could open the pcap file with Wireshark and inspect packets. We could apply filters and also directly follow TCP stream of OCSP protocol. We focused on HTTP carrying this OCSP data. This way we could see for example OCSP requests using POST and their contents.

Tshark traffic monitoring

We have determined how OCSP communication looks like and how the requests are structured, comparing theoretical sources, our sample observations and GET request analysis as described in Section 4.2. Then we decided to look at our daily traffic, filtering only OCSP requests (since we were interested only in the client-generated traffic).

As a tool, we have used tshark. We have composed several filters, at first capturing also unwanted packets or, on the contrary, not capturing complete OCSP traffic (a big step was composing a filter for GET request). Finally, we used the following tshark filter:

```
$ tshark -i network_interface -Y 'http contains "
   application/ocsp-request" or http.request.uri matches
   "(.)*/M..w(.)*" ' -T fields -E separator="," -e http.
   user_agent -e http.content_type -e http.request.method
    -e tcp.port -e http.host -e http.request.uri
```

Listing 6.2: *tshark* filter for OCSP requests

This filter relies on a fact that POST requests can be captured by the Content-Type header having the value "application/ocsp-request" and GET requests can be captured by http.request.uri matching regular expression "(.)*/M..w(.)*".

We were interested in the following data fields and printed them out as CSV:

- HTTP User-Agents,
- HTTP content type,
- HTTP request method,
- TCP port,
- HTTP host,
- HTTP request uri.

User-Agents provided us with information usable to identify a device and possibly also a browser or other application, such as e-mail client.

Content type field was empty with GET requests. Request method was either POST or GET. We have set the filter to capture these two request methods because this conforms to RFC 6960 [4] and actual implementation. TCP source port was different for every connection, therefore not interesting and unusable. TCP destination port was always 80 as HTTP is used to carry OCSP GET and POST requests. HTTP host represents OCSP responder queried. HTTP request URI was critical for identifying OCSP GET requests but consisting just from "/" or "/ocsp" for POSTs.

Sample traffic generated by Firefox 50.1.0, Chrome 55.0.2883.87 and Thunderbird 45.5.1 (in this order) on Ubuntu 16.04 follows:

```
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko
   /20100101 Firefox/50.0,application/ocsp-request,POST
   ,59322,80,ocsp.digicert.com,/

Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko
   /20100101 Firefox/50.0,application/ocsp-request,POST
   ,51192,80,ss.symcd.com,/
```

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML
   , like Gecko) Chrome/55.0.2883.87 Safari/537.36,,GET
   ,51404,80,ocsp.ws.symantec.com,/
   MHEwbzBNMEswSTAJBgUrDgMCGgUABBQ%2
   FYHKj6JjF6UBieQioTYpFsuEriQQUtnf6
   aUhHn1MS1cLqBzJ2B9GXBxkCEAm3Sf1%2FC0kWygVWVs%2
   F22YKiHjAcMBoGCSsGAQUFBzABBAQNMAsGCSsGAQUFBzABAQ\%3D
   \%3D

Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML
   , like Gecko) Chrome/55.0.2883.87 Safari/537.36,,GET
   ,51406,80,sh.symcd.com,/
   MHEwbzBNMEswSTAJBgUrDgMCGgUABBQNJg%2B66MGvVUAbA7bBb%2
   BmIp4Sc9QQUsm3j5BQPjDxzQqZamRrTFHW2htsCEBhp6ai4ShNaSOW
    HvIhcDsuiHjAcMBoGCSsGAQUFBzABBAQNMAsGCSsGAQUFBzABAQ%3
   D%3D

Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
   Thunderbird/45.5.1,application/ocsp-request,POST
   ,59800,80,ocsp.digicert.com,/

Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
   Thunderbird/45.5.1,application/ocsp-request,POST
   ,58156,80,clients1.google.com,/ocsp

Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101
   Thunderbird/45.5.1,application/ocsp-request,POST
   ,59800,80,ocsp.digicert.com,/
```

Listing 6.3: OCSP requests in sample traffic

From this small-scale experiment we could see that filter was composed right, the OCSP communication is really going on and the device identification can take place.

OS and application detection in normal traffic

We have composed the following *tshark* filter:

```
$ tshark -i eth2 -i eth3 -f 'tcp port 80' -R 'http
   contains "application/ocsp-request" or http.request.
   uri matches "(.)*/M..w(.)*" ' -T fields -E separator
   "," -e http.user_agent -e http.content_type -e http.
   request.method -e tcp.port -e http.host -e http.
   request.uri -a duration:3600
```

Listing 6.4: *tshark* filter for testing traffic

This command was run on testing traffic for an hour. The capture was limited just to TCP port 80 to conserve CPU. Captured traffic was written into file *privacy_all.txt*. This file has 5663 lines. 4425 were generated by POST, 1238 by GET requests.

Unique User-Agents and their count is in the file *privacy_unique.txt*. There were 91 unique user-agents.

The following are statistical information we were able to deduce from captured traffic. Table 6.1 contains User-Agent by groups and Table 6.2 shows a total number of different operating systems. We have decided to divide only MS Windows by versions since version distribution of Linux and Mac in the captured data wasn't interesting. The dictionaries used to infer the operating systems from captured values are in Appendix A.4.

Group name	OS	all	GET	POST
Windows NT	MS Windows	4048	7	4041
MS Crypto-API	MS Windows	1138	1138	0
X11	any	329	15	314
ocspd	Mac	46	45	1
Java	any	39	0	39
securityd	Mac	20	20	0
Macintosh	Mac	19	0	19
trustd	Mac	11	11	0
Mozilla/4.0 (Linux)	Linux	4	0	4
Mozilla/4.0 (Windows)	MS Windows	4	0	4
PPKHandler	unknown	3	0	3
empty	any	2	2	0

Table 6.1: OCSP request with various User-Agents

Just from access to a network and ability to capture outgoing OCSP traffic, a person is able to learn this many information. This is a privacy issue.

In the file *responders.txt*, we list and count OCSP responders visited. We can observe that most often visited are *clients1.google.com*,

Operating system	Number
Windows 7	2307
Windows 10	2230
Windows 8.1	334
Linux	333
Windows XP	241
Mac	96
Windows 8	70
unknown	44
Windows Vista	8

Table 6.2: Operating system distribution

ocsp.digicert.com and *??.symcd.com* (?? stands for two characters). This information can be used when someone wants to focus just on traffic going to these servers.

6.2 Measurement of latency issues

One of the issues of OCSP protocol was latency in communication. We wanted to do our own measurement of latency defined as the time difference between a request's timestamp and the timestamp of the corresponding response. However, this task couldn't have been successfully completed, because testing equipment produced very faulty output files. In this section, we provide at least the methodology and the explanation of our obstacles.

At first, the following *tshark* filter has been used for capturing all OCSP requests and responses (the capture was limited just to TCP port 80):

```
$ tshark -i any -f 'tcp port 80' -R 'http contains "
   application/ocsp-request" or http contains "
   application/ocsp-response" or http.request.uri matches
    "(.)*/M..w(.)*" ' -T fields -E separator="," -e frame
   .time -e http.content_type -e http.request.method -e
   ip.src -e ip.dst -a duration:60 > pairs
```

Listing 6.5: *tshark* filter for all OCSP communications

We have used source and destination IP address pair as a key connecting a request to its response. If there are more requests or responses with the same key, they are paired following the rule: "first with first, etc.", order being defined according to the timestamps.

We have used some bash scripts to preprocess the input. These can be found in Appendix A.3. There is also explained a C++ code computing the latency in the form of time differences between request and responses.

Our experiment was not successful because many requests or responses were missing or were not captured. For example, if out of 8 request-response pairs only requests number *1* and *3* and responses number *7* and *8* were captured, they were matched incorrectly. We were not able to distinguish this from a correct matching. Resulting time differences were hence incorrect and we could not deduce latency of OCSP traffic. However, in the file *latencies.txt* we provide computed values.

To the possible causes of faulty capture files belong:

- requests send before the start of the capture, with responses captured,
- responses received after the end of the capture, with requests captured,
- losses on the network,
- losses during *tshark* operation,
- incomplete requests or responses captured.

6.3 SHA-1 usage

This section describes how we have detected SHA-1 being used as a hash algorithm in OCSP request. Although we don't know about any possible malicious use of crafted colliding OCSP request, SHA-1 as an insecure algorithm should not be used.

Detecting SHA-1 in POST

We have used the following *tshark* filter to capture all OCSP requests carried by POST and to print their contents:

```
$ tshark -i network_interface -Y 'http contains "
   application/ocsp-request" ' -V > posts
```

Listing 6.6: *tshark* filter for OCSP POST request capture

We redirected output to a file and used the following bash command to cut out just request bodies:

```
$ sed -n '/Online/,/serialNumber/p' posts >
   resultSHA1post.txt
```

Output was again redirected to the file *resultSHA1post.txt*, which is included in this work. The last step was counting the number of *Algorithm Id* and comparing it to a number of *Algorithm Id: 1.3.14.3.2.26 (SHA-1)*, representing the SHA-1 algorithm.

```
$ grep 'Algorithm Id:' < resultSHA1post.txt | wc -l
$ grep 'Algorithm Id: 1.3.14.3.2.26 (SHA-1)' <
   resultSHA1post.txt | wc -l
```

The numbers matched (532 POST requests) and therefore we state that hash algorithm in every captured OCSP POST request has been SHA-1.

Detecting SHA-1 in GET

We have used the following *tshark* filter to capture all OCSP requests carried by GET and to print some of their value fields to the file *resultSHA1get.txt*, which is included in this work:

```
$ tshark -i network_interface -Y 'http.request.uri
   matches "(.)*/M..w(.)*" ' -T fields -E separator="," -
   e http.user_agent -e http.request.method -e http.host
   -e http.request.uri > resultSHA1get.txt
```

This work already explains, how is the request body encoded in *http.request.uri* field. The presence of SHA-1 algorithm is detected by looking for substring **BgUrDgMCGgUA** or **MAkGBSsOAwIa**.

```
$ grep -e BgUrDgMCGgUA < resultSHA1get.txt | wc -l
$ grep -e MAkGBSsOAwIa < resultSHA1get.txt | wc -l
```

There were 168 GET requests of the first type and 21 GET request of the second type, which is 189 GET request using SHA-1 in total.

This number matched the number of lines in the file (189 GET requests), therefore it is proven that every captured OCSP GET request has used the SHA-1 hash algorithm.

Presence of SHA-1 in encoded URL

There are two types of OCSP request structures. It is the result of freedom given in implementation.

The first type does not contain **[0] INTEGER 0** under second SEQUENCE.

The second type, which contains this fields, is generated by *ocspd*, *securityd* and *trustd*.

```
SEQUENCE {
  SEQUENCE {
    [0]
      INTEGER 0
    SEQUENCE {
      SEQUENCE {
        SEQUENCE {
          SEQUENCE {
            SEQUENCE {
               OBJECTIDENTIFIER 1.3.14.3.2.26 (id_sha1)
               NULL
            }
            OCTETSTRING
               f2e06af9858a1d8d709b4919237aa9b51a287e64
            OCTETSTRING 4
               add06161bbcf668b576f581b6bb621aba5a812f
            INTEGER 0x30c9a33fb239662b
          }
        }
      }
    }
  }
}
```

Listing 6.7: OCSP request body types

In the hex form of both types, `06 05 2b 0e 03 02 1a 05 00` stands for **OBJECTIDENTIFIER 1.3.14.3.2.26 NULL** and `2b 0e 03 02 1a` stands for **1.3.14.3.2.26** (SHA-1).

First type analysis

Sample GET request:

```
30 49 30 47 30 45 30 43 30 41 30 09 06 05 2b 0e 03 02 1a
05 00 04 14 f2 e0 6a f9 85 8a 1d 8d 70 9b 49 19 23 7a a9 b5
1a 28 7e 64 04 14 4a dd 06 16 1b bc f6 68 b5 76 f5 81 b6 bb
62 1a ba 5a 81 2f 02 08 30 c9 a3 3f b2 39 66 2b
```

We want to encode some part of this request which uniquely represents SHA-1. Because encoding in GET requests uses 6-tuples of bits, we can ignore first 12 bytes = 96 bits, because 96 is divisible by 6, leaving 16 characters. Then we can encode `06 05 2b 0e 03 02 1a 05 00` standing for **OBJECTIDENTIFIER 1.3.14.3.2.26 NULL**.

1. in binary: |00000110|00000101|00101011|00001110|00000011|00000010|00011010|00000101|00000000|
2. Base64 encoding converts it to characters
3. at first align bits to 6-tuples: |000001|100000|010100|101011|000011|100000|001100|000010|000110|100000|010100|000000|
4. then assign a character to each 6-tuple:
   |B|g|U|r|D|g|M|C|G|g|U|A|

So in SHA-1 GET request of the first type should be 16 characters and then this sequence: **BgUrDgMCGgUA**.

Second type analysis

Sample GET request:

```
30 56 30 54 a0 03 02 01 00 30 4d 30 4b 30 49 30 09 06 05
2b 0e 03 02 1a 05 00 04 14 df aa 12 e3 28 b1 09 41 93 e2 9f
42 82 ce 47 40 42 95 58 a3 04 14 b1 3e c3 69 03 f8 bf 47 01
d4 98 26 1a 08 02 ef 63 64 2b c3 02 10 0c 79 a9 44 b0 8c 11
95 20 92 61 5f e2 6b 1d 83
```

We want to encode some part of this request which uniquely represents SHA-1. Because encoding in GET requests uses 6-tuples of bits, we can ignore first 15 bytes = 120 bits, because 120 is divisible by 6, leaving 20 characters. Then we can encode `30 09 06 05 2b 0e 03 02 1a` standing for **SEQUENCE OBJECTIDENTIFIER 1.3.14.3.2.26**.

1. in binary: |00110000|00001001|00000110|00000101|00101011|00001110|00000011|00000010|00011010|
2. Base64 encoding converts it to characters
3. at first align bits to 6-tuples: |001100|000000|100100|000110|000001|010010|101100|001110|000000|110000|001000|011010|100000|010100|000000|
4. then assign a character to each 6-tuple:
   |M|A|k|G|B|S|s|O|A|w|I|a|

So in SHA-1 GET request of the second type should be 20 characters and then this sequence: **MAkGBSsOAwIa**.

7 Conclusion

The goal of this work was the network traffic analysis of OCSP protocol, which is being used to check the status of X.509 certificates. This work starts with an introduction to digital certificates and their revocation. Then we provide an analysis of OCSP protocol, its implementation, and structure of used messages. We see this as the main contribution of this work because as far as we know, OCSP was so far described only in informational Request for Comments (RFC) documents and its implementation was presented partially in various blogs and articles of respective companies.

We have validated means to capture OCSP traffic in a network, utilizing network monitoring tool *tshark*. This knowledge can be used in other research projects when examination of OCSP traffic is needed.

OCSP was designed to solve problems of older revocation status checking method of CRLs. However, it is not a perfect mechanism and has introduced new issues. Main of them are the breach of privacy, connection slow-down, and vulnerability to various types of attacks. In this work, we have demonstrated the ease of device identification with the help of OCSP traffic, provide a methodology to measure introduced latency and point out usage of an insecure hash algorithm of SHA-1.

We have shown that today in encrypted (HTTPS, TLS) communication there is unencrypted traffic of OCSP. This can be used by attackers to infer information about devices and applications on the network. However, this information cannot be considered accurate enough for monitoring of one's own network. It is because the volume of OCSP traffic is not big and cannot be deemed always true. For example, User-Agent can be easily changed by the user.

As stated before, the main contribution of this work is a usable knowledge base to OCSP and related area. Results of this work can be further used, possibly leading to implementation in some proof-of-concept open-source application. On the other hand, it presents some OCSP limitations, leaving authorities in the field of digital certificates with two possibilities: either address and correct its issues or propose a new approach, for examples short-term certificates.

Results of the thesis can be used in the research project named Research of Tools for Cyber Situational Awareness and Decision Support of CSIRT Teams in Protection of Critical Infrastructure [38].

Bibliography

[1] *An Overview of Public Key Infrastructures (PKI)*. [online]. Oct. 2016. URL: http://www.techotopia.com/index.php/An_Overview_of_Public_Key_Infrastructures_(PKI) [cited 2017-05-20].

[2] *Public Key Infrastructure*. [online]. URL: https://www.tutorialspoint.com/cryptography/public_key_infrastructure.htm [cited 2017-05-20].

[3] D. COOPER et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. [online]. RFC Editor, May 2008. URL: https://tools.ietf.org/html/rfc5280 [cited 2017-05-20].

[4] S. SANTESSON et al. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC 6960. [online]. RFC Editor, June 2013. URL: https://tools.ietf.org/html/rfc6960 [cited 2017-05-20].

[5] A. DEACON and R. HURST. *The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments*. RFC 5019. [online]. RFC Editor, Sept. 2007. URL: https://tools.ietf.org/html/rfc5019 [cited 2017-05-20].

[6] *ocsp - Online Certificate Status Protocol utility*. [online]. URL: https://www.openssl.org/docs/man1.0.2/apps/ocsp.html [cited 2017-05-20].

[7] *PKI Enhancements in J2SE 5*. [online]. URL: http://docs.oracle.com/javase/1.5.0/docs/guide/security/pki-tiger.html#OCSP [cited 2017-05-20].

[8] *ocspd(1) Mac OS X*. [online]. URL: https://developer.apple.com/legacy/library/documentation/Darwin/Reference/ManPages/man1/ocspd.1.html [cited 2017-05-20].

[9] *CryptoAPI*. [online]. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ms721572(v=vs.85).aspx#_security_cryptoapi_gly [cited 2017-05-20].

[10] *Cryptography API: Next Generation*. [online]. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa376210(v=vs.85).aspx [cited 2017-05-20].

[11] *Implementing an OCSP responder*. [online]. June 2009. URL: https://blogs.technet.microsoft.com/askds/2009/06/24/implementing-an-ocsp-responder-part-i-introducing-ocsp/ [cited 2017-05-20].

[12] *The AIA (Authority Information Access) extention*. [online]. Aug. 2014. URL: https://www.tbs-certificates.co.uk/FAQ/en/453.html [cited 2017-05-20].

[13] *How Certificate Revocation Works*. [online]. Mar. 2012. URL: https://technet.microsoft.com/en-us/library/ee619754(WS.10).aspx [cited 2017-05-20].

[14] *Network Security Services*. [online]. May 2016. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS [cited 2017-05-20].

[15] *NSS OCSP Brainstorming*. [online]. May 2007. URL: https://wiki.mozilla.org/NSS_OCSP_Brainstorming [cited 2017-05-20].

[16] L. ZHU, J. AMANN, and J. HEIDEMANN. "Measuring the Latency and Pervasiveness of TLS Certificate Revocation". In: *Passive and Active Measurement: 17th International Conference, PAM 2016, Heraklion, Greece, March 31 - April 1, 2016. Proceedings*. Ed. by T. KARAGIANNIS and X. DIMITROPOULOS. Cham: Springer International Publishing, 2016, pp. 16–29. ISBN: 978-3-319-30505-9.

[17] Wikipedia. *Extended Validation Certificate — Wikipedia, The Free Encyclopedia*. [online]. 2017. URL: https://en.wikipedia.org/

w/index.php?title=Extended_Validation_Certificate&oldid=781831991 [cited 2017-05-23].

[18] *Details on SSL/TLS certificate revocation mechanisms on iOS*. [online]. URL: https://web.archive.org/web/20121026094830/http://www.inmite.eu/en/blog/20120302-details-certificate-revocation-mechanisms-on-ios-iphone [cited 2017-05-20].

[19] A. CECIL. *A Summary of Network Traffic Monitoring and Analysis Techniques*. [online]. URL: http://www.cse.wustl.edu/~jain/cse567-06/ftp/net_monitoring.pdf [cited 2017-05-20].

[20] T. PORTER. *The Perils of Deep Packet Inspection*. [online]. Jan. 2005. URL: http://www.securityfocus.com/infocus/1817 [cited 2017-03-01].

[21] B. CLAISE, B. TRAMMELL, and P. AITKEN. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. [online]. RFC Editor, Sept. 2013. URL: https://tools.ietf.org/html/rfc7011 [cited 2017-05-20].

[22] *Wireshark*. [online]. URL: https://www.wireshark.org/ [cited 2017-05-20].

[23] Wikipedia. *User agent — Wikipedia, The Free Encyclopedia*. [online]. 2017. URL: https://en.wikipedia.org/w/index.php?title=User_agent&oldid=768827776 [cited 2017-03-06].

[24] B. KALISKI. *A Layman's Guide to a Subset of ASN.1, BER, and DER*. [online]. Nov. 1993. URL: http://luca.ntop.org/Teaching/Appunti/asn1.html [cited 2017-05-20].

[25] D. KEELER. *OCSP Stapling in Firefox*. [online]. July 2013. URL: https://blog.mozilla.org/security/2013/07/29/ocsp-stapling-in-firefox/ [cited 2017-05-20].

[26] M. ZWEERINK. *The current state of certificate revocation (CRLs, OCSP and OCSP Stapling)*. [online]. Jan. 2016. URL: https://www.

maikel.pro/blog/current-state-certificate-revocation-crls-ocsp/ [cited 2017-05-20].

[27] *HTTPS Everywhere*. [online]. URL: https://addons.mozilla.org/cs/firefox/addon/https-everywhere/ [cited 2017-05-20].

[28] *Revocation checking and Chrome's CRL*. [online]. Feb. 2012. URL: https://www.imperialviolet.org/2012/02/05/crlsets.html [cited 2017-05-20].

[29] M. GOODWIN. *Improving Revocation: OCSP Must-Staple and Short-lived Certificates*. [online]. Nov. 2015. URL: https://blog.mozilla.org/security/2015/11/23/improving-revocation-ocsp-must-staple-and-short-lived-certificates/ [cited 2017-05-20].

[30] M. PRINCE. *OCSP Stapling: How CloudFlare Just Made SSL 30% Faster*. [online]. Oct. 2012. URL: https://blog.cloudflare.com/ocsp-stapling-how-cloudflare-just-made-ssl-30/ [cited 2017-05-20].

[31] *Vulnerability Summary for CVE-2016-6304*. [online]. Sept. 2016. URL: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-6304 [cited 2017-05-20].

[32] *Announcing the first SHA1 collision*. [online]. Feb. 2017. URL: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html [cited 2017-05-20].

[33] *New version of the popular NGINX web server to support OCSP-stapling*. [online]. URL: https://www.nginx.com/press/globalsign-digicert-and-comodo-collaborate-nginx-improve-online/ [cited 2017-05-20].

[34] Wikipedia. *OCSP stapling — Wikipedia, The Free Encyclopedia*. [online]. 2016. URL: https://en.wikipedia.org/w/index.php?title=OCSP_stapling&oldid=726342235 [cited 2017-03-21].

[35] Y. PETTERSEN. *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*. RFC 6961. [online]. RFC Editor, June 2013. URL: https://tools.ietf.org/html/rfc6961 [cited 2017-05-20].

[36] P. HALLAM-BAKER. *X.509v3 Transport Layer Security (TLS) Feature Extension*. RFC 7633. [online]. RFC Editor, Oct. 2015. URL: https://tools.ietf.org/html/rfc7633 [cited 2017-05-20].

[37] *VirtualBox*. [online]. URL: https://www.virtualbox.org/ [cited 2017-05-20].

[38] *Výzkum nástrojů pro hodnocení kybernet. situace a podporu rozhodování CSIRT týmů při ochraně kritické infrastruktury (Crusoe)*. [online]. URL: https://www.muni.cz/vyzkum/projekty/35444 [cited 2017-05-20].

[39] *tshark - Dump and analyze network traffic*. [online]. URL: https://www.wireshark.org/docs/man-pages/tshark.html [cited 2017-05-20].

[40] *wireshark-filter - Wireshark filter syntax and reference*. [online]. URL: https://www.wireshark.org/docs/man-pages/wireshark-filter.html [cited 2017-05-20].

[41] *Bro - Windows version detection*. [online]. URL: https://www.bro.org/sphinx/_downloads/windows-version-detection.bro [cited 2017-05-20].

A Appendices

A.1 List of files

- *privacy_all.txt* – captured traffic in the experiment in Section 6.1,
- *privacy_unique.txt* – contents of *privacy_all.txt* after unique operation,
- *responders.txt* – list of OCSP responders visited in the experiment in Section 6.1,
- *compute_latency.cpp* – a C++ code used to process data in the experiment in Section 6.2,
- *latencies.txt* – computed latency values from the experiment in Section 6.2,
- *resultSHA1post.txt* – OCSP POST requests captured in the experiment in Section 6.3,
- *resultSHA1get.txt* – OCSP GET request captured in the experiment in Section 6.3.

A.2 Example of *tshark* filter

tshark is a program run from command-line. In manpages is a specification of option arguments [39]. Other manpages [40] explain how to set a filter for Wireshark/tshark. Most common option arguments are:

- **-r** = input file,
- **-Y** = display filter - applied before printing a decoded form of packets or writing packets to a file,
- **-E** = field print option - set an option controlling the printing of fields when **-T fields** is selected (-E separator="," can be used to gain CSV),

- **-T** = set the format of the output when viewing decoded packet data. **-T fields** = the values of fields specified with the -e option, in a form specified by the -E option,
- **-e** = add a field to the list of fields to display if **-T fields** is selected.

For example, a filter to capture all HTTP requests and responses and display captured user-agents and the content type of the packets would be:

```
$ tshark -i network_interface -Y "http.request or http.
   response" -T fields -E separator="," -e http.
   user_agent -e http.content_type
```

Listing A.1: Example of *tshark* filter

A.3 Processing of latency measurement

The following bash commands have been run to divide requests and responses to separate files:

```
$ grep -e 'application/ocsp-request' -e 'GET' pairs >
   requestsTmp
$ grep -e 'application/ocsp-response' pairs >
   responsesTmp
```

We have used the following bash script to preprocess the input providing us with 3-tuples of **seconds**, **microseconds** (in epoch time) and **IP address pair**:

```
while read line; do
date="$(cut -d, -f1-2 <<< "$line")"
microseconds="${date#*.}"
microseconds="${microseconds%% *}"
date="$(date -d "$date" +%s)"
ip1="$(cut -d, -f5 <<< "$line")"
ip2="$(cut -d, -f6 <<< "$line")"
echo "$date $microseconds $ip1,$ip2"
done < requestsTmp > requests

while read line; do
date="$(cut -d, -f1-2 <<< "$line")"
```

```
microseconds="${date#*.}"
microseconds="${microseconds%% *}"
date="$(date -d "$date" +%s)"
ip1="$(cut -d, -f5 <<< "$line")"
ip2="$(cut -d, -f6 <<< "$line")"
echo "$date $microseconds $ip2,$ip1"
done < responsesTmp > responses
```

A C++ code was then run, providing time differences as an output for further examination. Program, included in file *compute_latency.cpp*, groups requests (responses) according to their IP address pairs and sorts according to their timestamps. It then pairs requests to responses with the same IP address pair according to the rule: "first with first, etc.". The computed time difference is then printed.

A.4 Dictionaries for OS derivation

The following dictionaries have been used to infer OS from User-Agent. There are of course many other operating systems, and User-Agents can be used to infer also particular application (such as a browser). We have constructed dictionaries just for the scope of this work.

Table A.1 translates Microsoft-CryptoAPI number to OS Windows version. As the source, we have used Windows version detection script by Bro [41]. *x* represents number specifying OS release. The Microsoft-CryptoAPI number is visible in OCSP GET requests.

OS Windows from OCSP POST requests and also other operating systems can be deduced from translating a substring in the respective User-Agent, using Table A.2.

Microsoft-CryptoAPI number	OS Windows version
Microsoft-CryptoAPI/5.131.2195.*x*	Windows 2000
Microsoft-CryptoAPI/5.131.2600.*x*	Windows XP
Microsoft-CryptoAPI/5.131.3790.*x*	Windows XP x64 or Server 2003
Microsoft-CryptoAPI/6.0	Windows Vista or Server 2008
Microsoft-CryptoAPI/6.1	Windows 7 or Server 2008 R2
Microsoft-CryptoAPI/6.2	Windows 8 or Server 2012
Microsoft-CryptoAPI/6.3	Windows 8.1 or Server 2012 R2
Microsoft-CryptoAPI/6.4	Windows 10 Technical Preview
Microsoft-CryptoAPI/10.0	Windows 10

Table A.1: *Microsoft-CryptoAPI number* to *OS Windows version* dictionary

User-agent contains:	Operating system
Windows NT 5.0 or Windows 2000	Windows 2000
Windows NT 5.1 or Windows XP	Windows XP
Windows NT 5.2	Windows Server 2003 or XP x64 Edition
Windows NT 6.0	Windows Vista
Windows NT 6.1	Windows 7
Windows NT 6.2	Windows 8
Windows NT 6.3	Windows 8.1
Windows NT 10.0	Windows 10
Windows Phone 8.1	Windows Phone 8.1
WM 10.0	Windows Mobile 10
OpenBSD	Open BSD
SunOS	Sun OS
Linux or X11	Linux
Mac_PowerPC or Macintosh	Mac OS
QNX	QNX
BeOS	BeOS
OS/2	OS/2

Table A.2: *Substring of User-Agent* to *operating system* dictionary