POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master Thesis

Certificate Validation and Domain Impersonation

Advisors
prof. Antonio Lioy
prof.ssa Diana Berbecaru

Candidate:
Corrado Vecchio

Accademic year 2020-2021

Summary

Security of the World Wide Web ecosystem depends on the ability of web browsers of detecting revoked certificates. TLS protocol ensures a secure connection between two entities, but it could not be enough in case browsers accept connection with web server hosting revoked certificates.

In the first chapter of the work I present X.509 certificates and the entire SSL ecosystem, together with some works related to the problem of certificate validation and domain impersonation in the web PKI.

Secondly, I introduce Certificate Transparency and its positive aspect in the Web PKI and I also present how to validate an SCT embedded in a TLS certificate.

I also analyse a X.509 certificate dataset corresponding to the Alexa Top 1M Sites, downloading more than 400.000 certificates.

I also study the behaviour of 6 different web browsers on handling revocation information under different situations and operating systems. I surprisingly find out that browsers apply always a soft fail approach when revocation information are not available and some of them check revocation status of the entire certificates appearing in the chain only in presence of EV-certificates.

Finally I tests TLS implementations of some libraries that provide a command line utility for emulating a TLS client and establishing a TLS connection with web server belonging to the Alexa Top 1M list. Results show TLS implementations validate differently certificate chains and some of them do not check the revocation status.

Contents

Chapter 1

Introduction

TLS protocol has been designed to secure communications over networks and nowadays it is used to provide security over Internet. It allows Internet's users to identify the remote party they are connecting to by means the use of SSL certificates, which are signed by trusted Certification Authorities (CAs). CAs are the core of the Public Key Infrastructure (PKI) since they allow clients to validate SSL certificates with the reconstruction of a certificate chain rooted by a trusted CA.

The percentage of Internet users and the time they spend on Internet grew exponentially in the last years, both for private and business usage, along with the number of cyber criminals occurring during critical transactions. Many sensitive transactions are carried out every day on the Internet, through bank and e-commerce websites, and certificate validation process plays a fundamental role for identifying the server client is connecting to before the establishment of a secure connection. Theoretically modern web browsers must perform all possible checks before establishing a secure connection with a web server, but this study demonstrates that they often fail in validating certificates.

The event that sees DigiNotar CA as protagonist [1] is a striking example of what criminals can do having the control of the CA's private key: they have issued fraudulent certificates and used it to impersonate famous domain owners and steal confidential information to web users. It was the first time a CA's certificate has been removed from the browser trusted lists. This tragic event in the history of Internet teaches how important it is to revoke a certificate and advertise the revocation as soon as possible, when the private key is compromised. For the same reason, a full certificates validation process must be performed before a client establish a secure connection with a web server including certificates status checking to detect whether certificates appearing in the chain have been revoked.

A full and complete certificates validation process require to check: matching between certificate's subject name and server's hostname, being sure certificates appearing in the chain are not revoked and expired, making sure the certificate is not self-signed and validating signature over each certificate up to the root one. Berbecaru et al [2] introduce the certificate validation process starting from the presentation of a famous security incident related to certificate validation.

Browsers developers often apply a "*Soft-fail*" approach by deciding to trust certificates even when they are not able to load revocation information. It is a positive aspect for usability, but becomes a big issue for PKI security; for this reason a "*Hard-fail*" is advisable, in which browsers do not trust certificates when revocation information are not available by loosing in usability for users.

Liu et al[3] give an overview at certificate revocations in the Web' PKI, discovering that a considerable percentage of served certificates have been revoked and certificate revocation information require an high latency and bandwidth to be acquired. Furthermore, browsers often do not worry to check whether certificates are still valid or not.

Chung et al [4] perform a study to determinate whether all parties involved in PKI (certificate authorities, web server administrators and web browsers) are ready to support OCSP Must Staple.

Berbecaru et al [5], instead, implement a system which provide online certificate status services to final users together with a OCSP client API easily integrated into PKI compatible applications aiming at performing revocation checking through OCSP.

Amann et al [6] explore new security features which have been added to TLS, HTTPS and PKI over the past five years such as Certificate Transparency (CT), HTTP Strict Transport Security (HSTS) and HTTP Public Key Pinning (HPKP) headers, Certification Authority Authorization (CAA) and TLS Authentication (TLSA) DNS-based extensions and Signalling Cipher Suite Value (SCSV). In particular they investigate the usage of these technologies (focusing on Certificate Transparency and new TLS and HTTPS extensions), how are used in combination and which protection level is achieved.

Berbecaru et al [7] present MBS-OCSP, an improvement of CPC-OCSP system, based on Merkle hash trees and suitable for wireless environments allowing clients to cache some received information for future usages.

Wazan et al [8] analyse the behaviour of HTTPS interception products (proxies and anti-virus programs) in validating X.509 certificates. They also study how web browsers handle revocation information, focusing on the OCSP stapling mechanism.

Zulfiqar et al [9] perform an interesting measurement study of cryptographic strength and the assumption of revocation mechanisms in the X.509 certificates, by analysing OCSP stapling, RSA public key collisions and the strength of certificate serial numbers.

Roberts et al [10] presents a new classification of impersonation attack named target embedding (it belongs to "Subdomain Spoofing" class of attacks) which does not modify the impersonated domain but uses a subdomain of the actual domain (for example apple.com-signing.id embeds the target domain apple.com while actual domain is com-signing.id. Let's encrypt has released a certificate to this domain on 2018). Authors perform a user study in order to understand whether users are subjected to this kind of attack: results show that users are more vulnerable to target embedding attack than other ones like typosquatting, bitsquatting or combosquatting.

In this study I firstly present the structure of X.509 certificates along with the authorities involved in the certificate issuance process and the way for retrieving revocation information (CRL and OCSP). I also dedicate an entire chapter to Certificate Transparency, an open, global and monitoring system based on append-only public logs that collect certificates issued by CAs and it gives the possibility for monitoring each new entry and offers to domain owners a way for detecting fraudulent certificates issuance.

Since most of websites nowadays host a certificate issued by Let's Encrypt [11] (more that 265 million), a no profit certification authority that provide X.509 certificates at no charge, I secondly downloaded more than 400.000 certificates belonging to *Alexa Top 1 Million Sites* and I performed some analysis over the created dataset: how many certificates have been issued by Let's Encrypt? How many advertised certificates have been revoked? How many certificates expired?

Since there are not specific guidelines on how browsers should perform certificate validation process, browsers developers are responsible of implementing the process from sketch and they should not take care about some fundamental aspects in the validation process as checking revocation information. One goal of this study is to understand how web browsers perform certificate validation process before establishing a secure connection paying attention on how they handle revocation information. For this purpose I defined the platforms and the browsers to tests and I set up an experimental testbed with which I ran tests useful to evaluate the browsers behaviour.

Finally I introduce 3 three libraries which provide a command line utility for impersonating a TLS client and so allow to establish a TLS connection with web servers and I compare the results of certificates validation process performed by these 3 libraries when connecting to all websites listed in the *Alexa Top 1 Million Sites* list updated at the 26th of August.

Chapter 2

Background

2.1 PKC: Public Key Certificate

Public Key Certificate (PKC) is a data structure used to securely bind a public key to some attributes through the signature of an authority usually named Certification Authority (CA). Attributes, generally, are not defined and they are the one in interest for the transaction being protected with the PKC. PKCs are important to achieve non repudiation of a digital signature and they are the complementary component of a personal private key.

The step before the creation of PKC, is the generation of an asymmetric key pair:

- SK: Secret Key;
- PK: Public Key.

2.1.1 Certification architecture

Public key certification procedure involves the following entities:

- Certification Authority (CA): generates and (eventually) revokes PKCs, but also is responsible for indicating the revocation status of certificates it has issued;
- Registration Authority (RA): verifies the claimed identity and attributes and then, if everything goes well, authorizes issue of PKCs;
- Validation Authority (VA): authority delegated by the CA to timely revoke PKCs (it is not mandatory because its role can be assigned to the CA). Revocation is more critical than issuance and certificates must be immediately revoked when asked.

PKC generation process

Certification Authorities collect certificates into repositories together with the list of revoked one. One possible architecture, summarized in figure 2.1, for certificate generation provides:

1. Key generation: an asymmetric key pair is generated directly by the user or the company provides it in case of an employee;
2. Key storing and forwarding: after user generates the key pair, he securely stores the private key (SK) in a place controlled only by him and sends the public one, together with an identifier (ID), to the CA;

3. Verification: every time a CA receives a request for issuing a certificate, it must verify the validity of the claimed identity and attributes. For this reason, the user needs to physically meet someone of the registration authority in order to proof the claimed identity and attributes. If everything goes well, RA confirm the identity and sends a request to CA for binding the ID and attributes with the given PK;
4. Issuance: CA is in charge of issuing the certificate (associating the ID with the PK) and publishes it over public repositories.

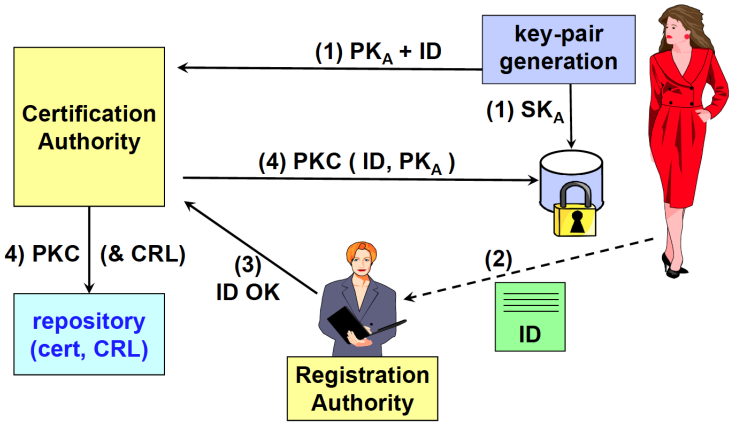


Figure 2.1: Certificate generation steps

PKC contains information which allows to uniquely associate a cryptographic key to an entity which has the control of the corresponding private key. The binding is guaranteed by a Trusted Third Party (TTP), usually called Certification Authority (CA), which digitally signs each certificate. When creating a certificate, the liability may be limited to specific applications or purpose (as specified in the CA's certification policies).

2.2 X.509 Certificates

X.509 standard has been defined by the International Telecommunications Union's (ITU-T) and nowadays is the most widely used for PKCs. X.509 standard appears for the first time in 1988 as part of the X.500 directory recommendation, and certificates format belonging to this standard is the version 1 (v1). In 1993 two fields have been added to the native X.500 standard leading to the version 2 (v2) format. Although native format has been revised with the release of second version, it was not enough yet: certificate formats (v1 and v2) were not able to meet requirements experience had proven necessary, such as the absence of enough fields for carrying information.

In order to meet requirements implementation experience had proven necessary, International Telecommunications Union's (ITU-T) modifies for the third time the standard realising the X.509 version 3 (v3) certificate format which extends the already existent v2 format by adding additional extension fields. Some extension field types have been defined into the v3 format as "standard extensions" (such as certification path constraints, key attribute information, additional subject identification information and policy information) while others may be defined and registered by any kind of organization or community depending on their needs.

Certificate Policy (CP) and Certificate Practice Statements (CPS) Certificate Policy (CP) is a set of rules that indicates the applicability of a PKC to a particular community and/or class of application, which have in common some security requirements. Certificate Practice Statements (CPS), instead, is a statement of the best practices employed by a CA in issuing PKC (how certificate policies have been implemented). CP specifies minimum requirements CAs have to respect when issuing a certificate (can be followed by many CAs) while CPS contains implementation details and it is typical of a single CA. RFC 3647 [12] defines a framework for writing certificate policies (CP) and certification practice statements (CPS).

Certification paths In order to establish a TLS connection among a client and a web server, the standard requires client validates server certificates. In case CA public key that signed the server certificate is not presented in the client root CA list (assured CAs), one or many additional certificates are needed, to get issuer public key. Generally it is needed a chain of multiple certificates composed as follow: an end entity certificate signed by one CA and zero or more additional certificates of CAs signed by other CAs. These chain are fundamental for validating an end entity certificate, because a normal client (e.g browser) holds only a list of limited numbered of trusted CA public keys. An example of a certificate chain is showed in figure 2.2: to validate the end certificate www.polito.it, 3 additional CAs certificates are needed:

- GEANT OV RSA CA 4: it is the CA certificate of the end entity certificate issuer;
- USERTrust RSA Certification Authority: it is the CA certificate of the "GEANT OV RSA CA 4" certificate issuer;
- Sectigo (AAA): it is the root CA certificate and it is the certificate of "USERTrust RSA Certification Authority" certificate issuer.

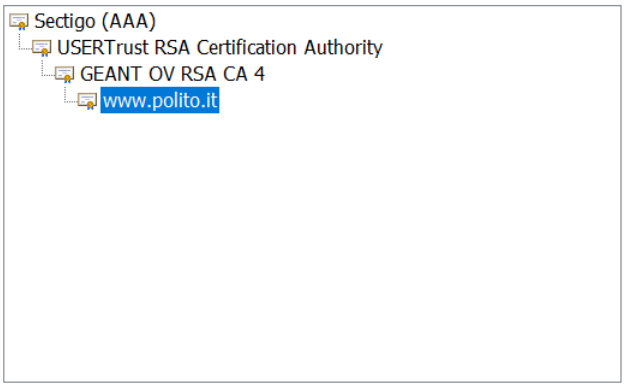


Figure 2.2: Certificate chain for the end certificate www.polito.it

One of possible CAs configuration, which allows public key users in finding certification paths, is presented in RFC-1422 [13]. According to this standard, there is an inflexible hierarchical structure made up of 3 types of PEM certification authorities:

- Internet Policy Registration Authority (IPRA): it operates as root of the PEM hierarchy, so that all certificate paths have IPRA as root. It issues certificates only for PCAs, which are the certification authorities of the next level;
- Policy Certification Authorities (PCAs): they represent the level 2 of the hierarchy, and each PCA is certified by IPRA. Each PCA can define and publish a list of its own policies related to different CA and user needs. In this way, distinct PCAs with different policies satisfy different user needs; For example a high-assurance PCA has more stringent policies for satisfying legally requirements while mid-level assurance policies, for example, may satisfy commercial organizations or electronic mail needs;
- Certification Authorities (CAs): they represent the level 3 of the hierarchy and can be also present in lower level.They are, for example, specific units, specific geographic areas, or specific organizations.

This CA configuration is related with the usage of X.509 v1 certificate format that has several restrictions, summarized as follow:

- Restricted flexibility due to hierarchical infrastructure: all certification paths root at IPRA;
- Name subordination rule: it requires that a CA can only issue certificates for entities whose names are subordinate to the name of the CA itself. This rule restricts the names of a CA's subjects;

- Usage of PCA concept: individual PCAs must be known to establish whether a chain can be accepted or not.

Certificate extensions added to X.509 v3 certificate format, allow to ensure most of the requirements addressed by RFC-1422 [13] without any kind of restriction for the CA structures used. In general, certificate extensions obviate the need for name subordination rule and PCAs defining a more flexible architecture:

- Certification paths root is a public key of a CA present in a user's trust list instead of being always rooted by IPRA;
- Name constraints extension allow to impose the name subordination rule. This extension is not required, but optional;
- Automation of certification path processing: PCA concept is achieved by means of policy extensions and policy mappings which also increase degree of automation. In this way, clients are able to determinate whether a certification path can be acceptable looking at the certificate content itself instead of a priori knowledge of PCAs.

X.509 v3 certificate format allows to distinguish whether the subject of a certificate is a CA or an end entity by means of an extension. CA certificates, according to RFC-5280 [14], may be also divided into three classes: cross-certificates, self-issued certificates and self signed certificates:

- Cross-certificates: CA certificates in which the issuer and the subject are different, and certificates describes a trust relationship among the two CAs;
- Self-issued certificates: CA certificates in which the subject and the issuer are the same entity;
- Self-signed certificates: self-issued certificates where the digital signature may be verified by means of the public key present into the certificate, and they are used to bind public keys for use to begin certification paths. Root CA certificates are self-signed certificates.

2.2.1 X.509 basic fields

A public key certificate is a structure of 3 required fields

```
Certificate ::= SEQUENCE {
    tbsCertificate TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue BIT STRING }
```

Listing 2.1: X.509 basic fields

TBSCertificate

The tbsCertificate field contains some certificate information such as version of the certificate, issuer, serial number, validity, subject, public key and some information about the algorithm used for signing the certificate.

Subject Subject field of a X509 certificate gives information about the organization related to the public key stored into the subject public key field. Information related to the subject may be presento into the subject field and optionally they can appear also inside the subjectAlternativeName extension:

- In case the certificate subject is a CA, content of issuer's certificate subject field must match the contents of issuer field in all certificates issued by that CA;

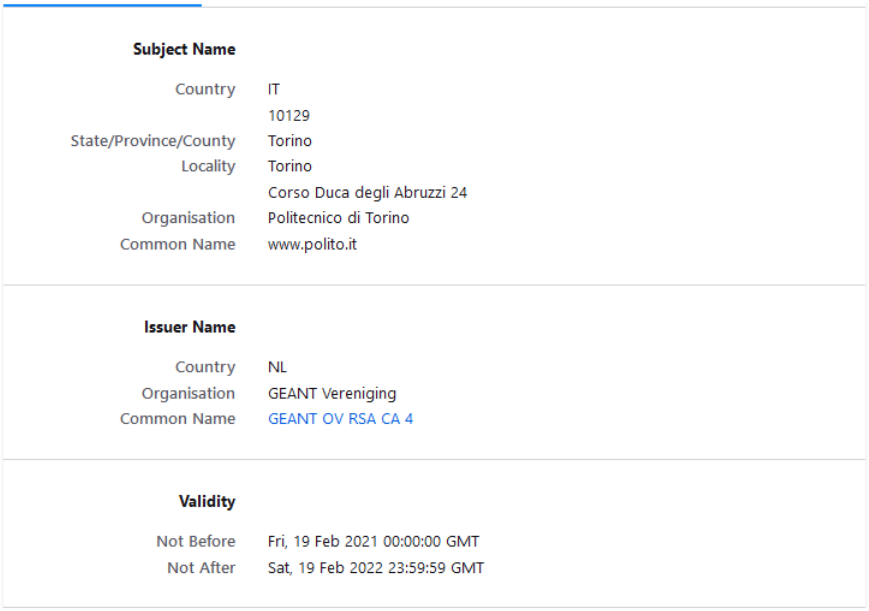


Figure 2.3: Subject name, issuer name and validity of certificate hosted by `www.polito.it`

- In presence of a certificates whose subject is a CRL issuer, information contained into the subject field must match the value of the issuer field in all CRLs issued by that CRL issuer;
- In case the subject information appears only in the subjectAltName extension, subject name field must be empty and the subjectAltName extension must be set as critical.

Into the certificate hosted by `www.polito.it` 2.3, "Subject" field lists information related to Politecnico di Torino: country, state/province/country, locality, organization and common name.

Issuer Information contained into Issuer field of a X.509 certificate identify the organization that has signed and issued the certificate. Issuer of polito certificate in 2.3 is "Geant Vereniging", a Dutch CA whose common name is "Geant OV RSA CA 4".

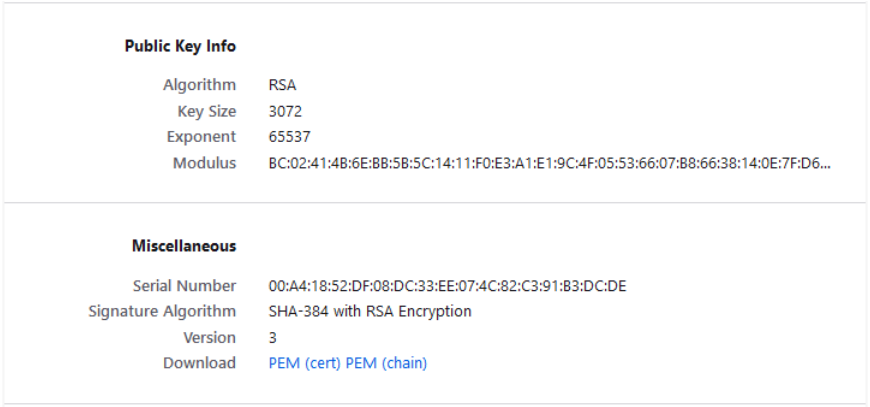
Validity Validity fields specifies the time interval during which the certificate is considered valid. It is composed of two dates:

- NotBefore: date and time from which the validity period of a X.509 certificate starts;
- NotAfter: date and time until the X.509 certificate is still valid.

Polito certificate in 2.3 is valid starting from the date specified in "Not Before" field (Fri, 19 Feb 2021 00:00:00) until the one contained in "Not After" field (Sat, 19 Feb 2022 23:59:59).

Public Key Info Public key info's aim is to store the public key and the algorithm used for its generation (e.g. RSA, Diffie-Hellman, DSA). Key associated with Polito certificate, in figure 2.4, is 3072 bits long and it has been generated using RSA algorithm with 65537 as public exponent. Modulus entry contains the public key.

Serial Number Serial number field identifies uniquely a X.509 certificate and it is always a positive and unique integer assigned directly by the issuing CA. Serial number of the certificate in 2.4 is 00:A4:18:52:DF:08:DC:33:EE:07:4C:82:C3:91:B3:DC:DE.



Public Key Info

Algorithm	RSA
Key Size	3072
Exponent	65537
Modulus	BC:02:41:4B:6E:BB:5B:5C:14:11:F0:E3:A1:E1:9C:4F:05:53:66:07:B8:66:38:14:0E:7F:D6...

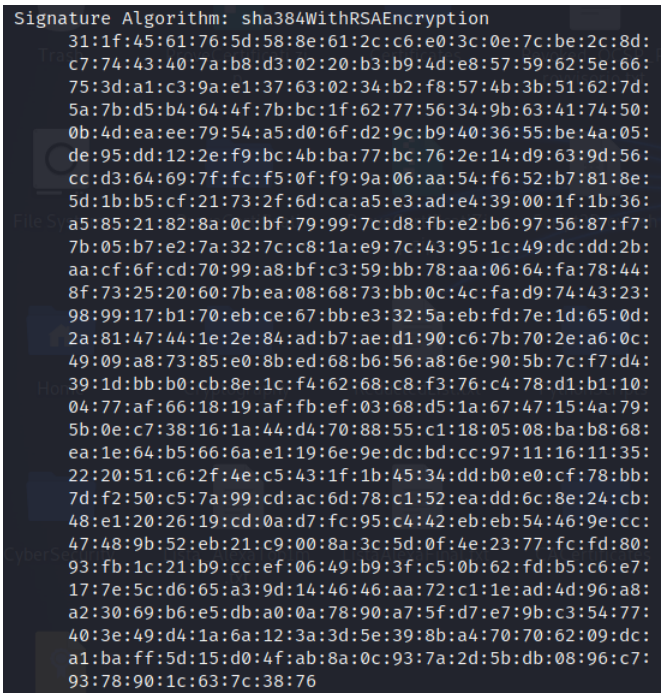
Miscellaneous

Serial Number	00:A4:18:52:DF:08:DC:33:EE:07:4C:82:C3:91:B3:DC:DE
Signature Algorithm	SHA-384 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)

Figure 2.4: Public key info, serial number, signature algorithm and version of certificate hosted by www.polito.it

Version Version field specifies the version of the X.509 certificate. In presence, at least, of an extension, version must be 3. Polito certificate in 2.4 has the version field set to 3.

Signature Signature field contains the identifier of the algorithm used by the CA to sign the certificate. String which identifies the algorithm used by the CA for signing polito certificate, showed in 2.4, is SHA-384 with RSA encryption.



```
Signature Algorithm: sha384WithRSAEncryption
     31:1f:45:61:76:5d:58:8e:61:2c:c6:e0:3c:0e:7c:be:2c:8d:
     c7:74:43:40:7a:b8:d3:02:20:b3:b9:4d:e8:57:59:62:5e:66:
     75:3d:a1:c3:9a:e1:37:63:02:34:b2:f8:57:4b:3b:51:62:7d:
     5a:7b:d5:b4:64:4f:7b:bc:1f:62:77:56:34:9b:63:41:74:50:
     0b:4d:ea:ee:79:54:a5:d0:6f:d2:9c:b9:40:36:55:be:4a:05:
     de:95:dd:12:2e:f9:bc:4b:ba:77:bc:76:2e:14:d9:63:9d:56:
     cc:d3:64:69:7f:fc:f5:0f:f9:9a:06:aa:54:f6:52:b7:81:8e:
     5d:1b:b5:cf:21:73:2f:6d:ca:a5:e3:ad:e4:39:00:1f:1b:36:
     a5:85:21:82:8a:0c:bf:79:99:7c:d8:fb:e2:b6:97:56:87:f7:
     7b:05:b7:e2:7a:32:7c:c8:1a:e9:7c:43:95:1c:49:dc:dd:2b:
     aa:cf:6f:cd:70:99:a8:bf:c3:59:bb:78:aa:06:64:fa:78:44:
     8f:73:25:20:60:7b:ea:08:68:73:bb:0c:4c:fa:d9:74:43:23:
     98:99:17:b1:70:eb:ce:67:bb:e3:32:5a:eb:fd:7e:1d:65:0d:
     2a:81:47:44:1e:2e:84:ad:b7:ae:d1:90:c6:7b:70:2e:a6:0c:
     49:09:a8:73:85:e0:8b:ed:68:b6:56:a8:6e:90:5b:7c:f7:d4:
     39:1d:bb:b0:cb:8e:1c:f4:62:68:c8:f3:76:c4:78:d1:b1:10:
     04:77:af:66:18:19:af:fb:ef:03:68:d5:1a:67:47:15:4a:79:
     5b:0e:c7:38:16:1a:44:d4:70:88:55:c1:18:05:08:ba:b8:68:
     ea:1e:64:b5:66:6a:e1:19:6e:9e:dc:bd:cc:97:11:16:11:35:
     22:20:51:c6:2f:4e:c5:43:1f:1b:45:34:dd:b0:e0:cf:78:bb:
     7d:f2:50:c5:7a:99:cd:ac:6d:78:c1:52:ea:dd:6c:8e:24:cb:
     48:e1:20:26:19:cd:0a:d7:fc:95:c4:42:eb:eb:54:46:9e:cc:
     47:48:9b:52:eb:21:c9:00:8a:3c:5d:0f:4e:23:77:fc:fd:80:
     93:fb:1c:21:b9:cc:ef:06:49:b9:3f:c5:0b:62:fd:b5:c6:e7:
     17:7e:5c:d6:65:a3:9d:14:46:46:aa:72:c1:1e:ad:4d:96:a8:
     a2:30:69:b6:e5:db:a0:0a:78:90:a7:5f:d7:e7:9b:c3:54:77:
     40:3e:49:d4:1a:6a:12:3a:3d:5e:39:8b:a4:70:70:62:09:dc:
     a1:ba:ff:5d:15:d0:4f:ab:8a:0c:93:7a:2d:5b:db:08:96:c7:
     93:78:90:1c:63:7c:38:76
```

Figure 2.5: Algorithm identifier and signature computed over the certificate hosted by www.polito.it

Signature algorithm

The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the CA to sign the certificate as the one contained in the TBScertificate sequence.

Signature value

The signatureValue field contains the digital signature computed over the TBSCertificate sequence of information. With the signature generated over the TBS structure, CA certifies the validity of information contained inside the TBS sequence: it attests the association among the public key material and the subject of the certificate.

Figure 2.5 shows that CA uses SHA 384 with RSA Encryption for computing the signature over the TBS sequence.

2.2.2 X.509 Certificate extensions

V3 version of X.509 certificates support the extension field, which is a sequence of one or more certificate extensions defined for associating additional information and attributes with X.509 basic fields. They can be:

- Public: ones defined inside the RFC-5280 [14] and consequently made public;
- Private: ones unique for a certain user community.

Each certificate extension can be defined as critical or non critical: during certificate verification process, any system based on certificates usage, must reject a certificate in case it is not able to recognize a critical extension or it is not able to process the critical extension content. Otherwise, a certificate-using system may ignore a non critical extension in case it has not been recognized.

Public extensions

Extensions belonging to this category are divided into 4 classes 2.1:

- Key and policy information: additional information about the key being certified and the certification policies followed in certifying the key;
- Certificate subject and certificate issuer attributes: additional attributes related to certificate subject and issuer;
- Certificate path constraints: constraints about the sequence of certification authorities;
- CRL distribution points: places where to download CRLs.

Authority key identifier (AKI) Authority key identifier extension is a way for identifying the public key correspondent to the private one used to sign a certificate because one CA can use two or multiple different keys for issuing certificates (e.g. to guarantee low and high assurance). The identification of the key used by the CA to sign the certificate may be based on the key identifier (the subject key identifier in the issuer's certificate) or on the issuer name - serial number pair. Since AKI extension facilitates certification path construction (during chain validation process), it must be present in all certificates issued by conforming CAs, excluding self-signed certificates where AKI extension can be empty since it would be equal to the subject key identifier extension. It is usually marked as non critical.

Autority Key Identifier extension of certificate hosted by `www.polito.it` 2.6 contains 6F:1D:35:49:10:6C:32:FA:59:A0:9E:BC:8A:E8:1F:95:BE:71:7A:0C as key identifier for the public key used for signing the certificate.

Table 2.1: Public extensions X.509 v3 certificates

Public Extensions			
Key and policy information	**Certificate subject and certificate issuer attributes**	**Certificate path constraints**	**CRL distribution points**
Authority key identifier	Subject alternative name	Basic constraints	CRL distribution points
Subject key identifier	Issuer alternative name	Name constraints	Freshest CRL
Key usage	Subject directory attributes	Policy constraints	
Private key usage			
Certificate policies			
Policy mappings			

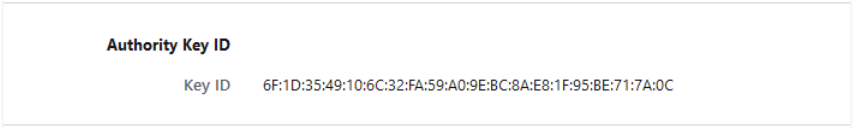


Figure 2.6: Authority Key Identifier extension of certificate hosted by www.polito.it

Subject Key Identifier Subject key identifier extension is a way for identifying certificates that contain a specific public key. Since this extension easies certification path construction (during chain validation process), it must be present in all conforming CA certificates (ones with basic constraint extension set to true). This extension facilitates certificate path construction during validation process and must be present in all conforming CA certificates The value of subject Key identifier extension is derived from public key contained into the certificates and RFC-5280 [14] lists some ways for computing key identifier starting from public key differently for CA and end-entity certificates.

Subject Key Identifier extension of certificate hosted by www.polito.it 2.7 contains 97:32:21:07:E2:BD:3B:DD:C9:DE:66:6C:35:6C:15:26:4E:9C:9F:9C as key identifier for the public key contained into the certificate itself.

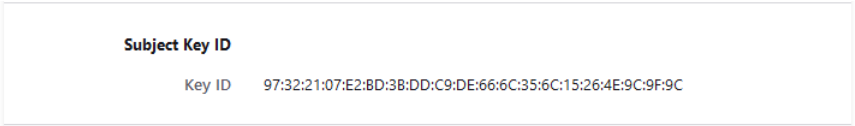


Figure 2.7: Subject Key Identifier extension of certificate hosted by www.polito.it

Key usages (KU) Key usage extension specifies the purpose (e.g. signature, encipherment, certificate signing) for which the key included in the certificate can be used. This extension can be marked as critical or non critical; in case the extension is marked as critical, the certificate can be used only for the purposes defined in the KU extension. Key usage extension can assume one or more of the following values:

- Digital Signature: subject public key is be used to verify digital signatures. It is valid both for CA certificates and user ones;
- Non Repudiation: subject public key is used to provide non-repudiation service. It is valid only for user certificates;
- Key Encipherment: subject public key is used for enciphering other keys. It is valid only for user certificates;
- Data Encipherment: subject public key is used for directly enciphering raw information without the use of an auxiliary symmetric cipher. It is rarely used because asymmetric encryption is a slow procedure;
- Key Agreement: subject public key is used for key agreement (e.g. DH parameter);
- Key Certificate Signature: subject public key is used for verifying signature on public key certificates. It is valid only for CA certificates;
- CRL Sign: subject public key is used for verifying signature on certificate revocation lists and it is valid only for CA certificates.
- Encipher Only: when keyAgreement bit is set, the subject public key may be used only for enciphering data while performing key agreement;
- Decipher Only: when keyAgreement bit is set, the subject public key may be used only for deciphering data while performing key agreement.

EncipherOnly and decipherOnly meaning is undefined in absence of keyAgreement bit. The combination of several values for the key usage extension limits context in which the certificate can be used.

Key usage extension of certificate hosted by www.polito.it 2.8 contains Digital Signature and Key Encipherment values. It means that the public key contained into the certificate can be used only for the two indicated purposes and it is also marked as critical (showed by the ! next to Key Usages text).

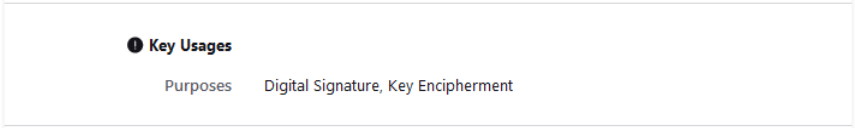


Figure 2.8: Key Usages extension of certificate hosted by www.polito.it

Private key usage period Private key usage period extension defines the usage period of the private key. This extension is always non critical and the usage is discouraged because the user itself should decide when change its private key. It is normally used for military scopes.

Certificate policies Certificate policies extension is a sequence of one or multiple policy information terms, each one identified by means of a unique OID and optionally contains a qualifier. This extension can assume different values for CA certificates and end-entity certificates. For end-entity certificates, this extension is useful for listing policies followed by CA during the certificate issuance and the purposes for which the certificate can be used. In CA certificates, this extension limits the policies for certification paths construction that include the certificate containing the extension. In case a CA has no interest in limiting the set of policies for certification paths construction, it can set the extension with the *anyPolicy* value (OID 2529320). Certificate policies extension can be marked as critical or non critical, and in case it has been marked as critical it must be rejected whether a path validation software is not able to interpret its content.

Certificate policies extension of certificate hosted by www.polito.it 2.9 lists 3 policies:

- Practices Statement (1.3.6.1.5.5.7.2.1): practices followed by CA in issuing and managing the certificate;
- Certificate Type (2.23.140.1.2.2): Organization Validation certificate.

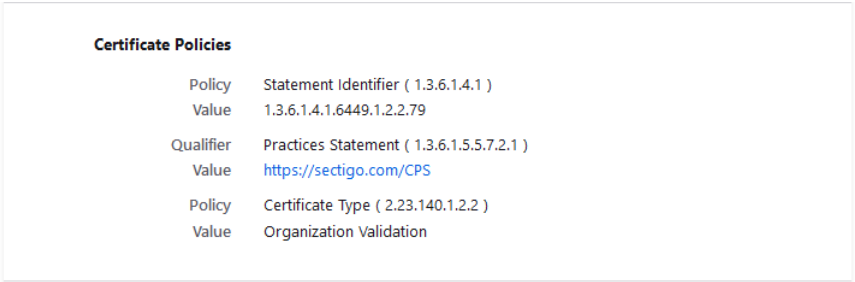


Figure 2.9: Certificate Policies extension of certificate hosted by www.polito.it

Policy mappings Policy mappings extension is present only in CA certificates and it indicates the correspondence of policies among different certification domain. It is a non critical extension.

Subject alternative name Subject alternative name extension provides different formats to identify the owner of a certificate such as e-mail address, IP address and URL. When the subject name field of a certificate is empty, the subject alternative name extension must be present and marked as critical; while in case the subject name field is non empty, the subject alternative name extension should be marked as non critical. Subject Alternative Name extension of certificate hosted by `www.polito.it` 2.10 lists 3 alternative subject names: `www.polito.it`, `polito.it` and `wwwtest.polito.it`.

Figure 2.10: Subject Alternative Name extension of certificate hosted by `www.polito.it`

Issuer alternative name Issuer alternative name extension allows to use different formalism to identify the CA that issued a certificate. This extension is always marked as critical in case the field issuer name of the certificate is empty; otherwise it should be marked as non critical. The types of alternative names are the same for Subject Alternative Name extension, defined in [14] in section 4.2.1.7.

Certificate hosted by `www.polito.it` does not contain Issuer Alternative Name extension.

Subject directory attributes Subject directory attributes extension allows to store identification attributes (e.g. nationality) of the subject. It is suggested to mark it as non critical.

Basic constraints Basic constraint extension allow to identify whether the subject of the certificate is a CA and also the maximum number of non-self issued intermediate CA certificate that may follow this certificate in a valid certification path. Basic constraint extension value is related with the content of key usage extension: if the basic constrain extension is not set to true, the key usage extension must not contain the Key Certificate Sign entry. In all CA certificates which use the contained public key for validating signature over end-entity certificates, the basic constraint extension must appear marked as critical. While it can be marked as non critical in such CA certificates containing a public key used exclusively for alternative purposes (e.g. validating digital signature on CRL or key management) other than validating digital signature. Finally in end entity certificates this extension can be marked as critical or non critical. Basic Constraint extension of certificate hosted by `www.polito.it` 2.11 has been set to false and it has been marked as critical, as warmly suggested by RFC-5280 [14]; while figure 2.12 shows basic constraint extension of the polito issuer's certificate: since it is a CA certificate, the extension is marked as critical and the value is set to true.

Figure 2.11: Basic Constraint extension of certificate hosted by `www.polito.it`

Name constraints Name constraints extension is valid only for CA certificates and it allows to restrict the possible values assignable to subject field of the certificates will be issued by CA in the form of excluded or permitted names.

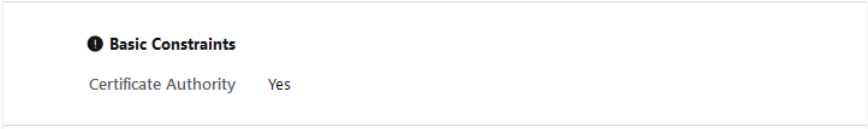


Figure 2.12: Basic Constraint extension of www.polito.it issuer's certificate

Policy constraints Policy constraints extension is meaningful in certificates issued to CAs (not in end entity certificates) and it affects path validation process in two ways: prohibiting policy mappings or requiring that each certificate in a path contain a certain policy identifier. This extension can be critical or non critical.

Extended key usage Extended key usage extension extends the purposes for which the public key bound in a certificate can be used, additionally to the ones present in key usage extension. This extension can be present only in end-entity certificates and ca be critical or non critical. A certificate can lists both a key usage and extended key usage extension and in this case they must be processed separately; the certificate can be used only for a purpose congruous with the two extensions. In absence of purpose consistent among the two extension, the certificate must not be used for any purpose. Extended key usage extension of certificate hosted by www.polito.it 2.13 lists two other (Server Authentication, Client Authentication) purposes for which the certificate can be used, in addition to the ones already present in the key usage extension.

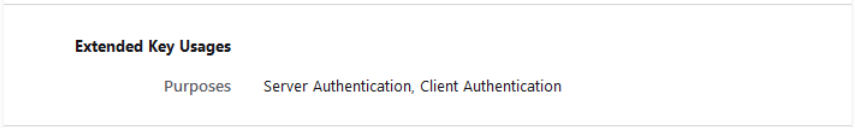


Figure 2.13: Extended key usage extension of certificate hosted by www.polito.it

CRL distribution points CRL distribution points extension lists one ore more distribution points for retrieving CRL information. This extension should appear as non critical. CRL distribution points extension of certificate hosted by www.polito.it 2.14 contains one distribution point, from which a CRL can be downloaded.

Inhibit anyPolicy Inhibit anyPolicy extension is valid only for CA certificates. It is used for indicating that the particular anyPolicy OID (2.5.29.32.0) is not considered an explicit match for other certificates policies. It specifies the number of additional CA certificates that may appear in the path before anyPolicy is no longer permitted.

Freshest CRL (Delta CRL Distribution Point) Freshest CRL extension is used to provide an additional CRL distribution point for downloading information about any certificates revoked since the last update to the full CRL. It must be marked as non critical so that the application is free to decide retrieving a delta CRL or the full one.

Private extensions

Private extensions are always extensions but they have been defined only for a specific user community and are meaningless outside context have been defined into. RFC 5280 [14], for example, defines 3 private extensions 2.2 for internet community:

- Subject information access: it provides a method (e.g. HTTP or LDAP) for accessing information and services offered by the subject in which the extension appears, by providing the format and the location. It is normally marked as non critical;

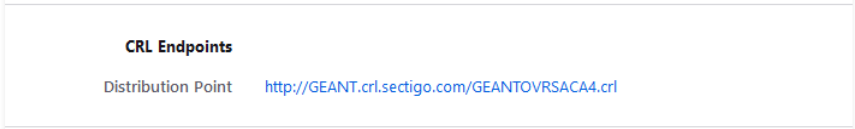


Figure 2.14: CRL distribution points extension of certificate hosted by `www.polito.it`

- Authority information access (AIA): it indicates how to access information and services offered by the issuer of the certificate in which the extension appears. Among these services there are:
  - certStatus: it allows to retrieve information about how to get the certificate's issuer (CA issuer access method);
  - caPolicy: it allows to retrieve policies followed by the CA;
  - certRetrieval: it allows to get the certificate of the CA itself;
  - caCerts: it allows to get the list of all certificates issued by the CA.

  This extension can be critical or non critical, but normally is marked as non critical.
- CA information access: it is valid only for CA certificates and it allows to get services offered by the CA itself. The possible services are the same of AIA extension and it also can be marked as critical or non critical (normally it is a non critical one).

Table 2.2: Private extensions X.509 v3 certificates defined for Internet community in RFC 5280[14]

Private extensions
Subject information access
Authority information access
CA information access

2.3 Certificate revocation

Public Key Certificates are issued by Certification Authorities for a limited period of time and CAs must renew them before they expire. This is what happen in the best of cases, when no unpleasant situations affect certificate life cycle. Nevertheless, it may be necessary to invalidate a certificate before its natural expiration due to different possible reasons such as a compromised private key or the usage of a weak algorithm for key pair generation. In case the certificate is not revoked, or it is not revoked in time, it affects the security of web ecosystem because someone else can impersonate the server's identity and can record every confidential information exchanged between any client and the "impersonated server"

When a client sets up a TLS connection with a web-server, the server submits a certificates chain embedded in the TLS handshake. Any certificate-usage system, for example a web browser, must validate the chain sent by the server: it must be sure that no certificates in the chain have been revoked. The publication of revocation information happens by means of two protocols: Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP).

2.3.1 Certificate Revocation List (CRL)

Checking revocation status by means of CRLs is the first way adopted for detecting revoked certificate, standardized in [14]. CRL structure contains 3 elements 2.2: a list of certificates (revoked certificates), the identifier of the algorithm used for computing the signature over the tbsCertList and the signature computed over the tbsCertList.

```
CertificateList ::= SEQUENCE{
    tbsCertList TBSCertList,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue BitString }
```

Listing 2.2: Structure of a CRL, defined in RFC-5280 [14]

The list presents inside CRL structure is a sequence of serial number, revocation timestamp and revocation reason of revoked certificates. Each certificate issued by a CA contains the url, inside the CRL distribution points extension, from which download the CRL. Retrieving revocation information about a certificate by means of CRL requires to download the CRL from the url present in the CRL distribution points extension, and verify the presence of the certificate's serial number inside the list. As for X.509 certificates, CRLs are valid for a limited amount of time. They are valid during the time interval between *thisUpdate* and *nextUpdate* tbsCertList fields. *This update* field contains the date in which the CRL has been issued while *next update* the date in which the CRL newly version will be issued even in absence of new revoked certificates. Clients are able to cache CRLs to reduce time needed for retrieving revocation information (and so improve performance) but they must be careful on download the newly versions of CRLs once they expire.

Picture 2.15 shows the fields of CRL downloaded from the url present in CRL distribution points extension of certificate hosted by `www.polito.it`. The CRL has been issued on the 3 of November and it expires on the 10 of November. Picture 2.16, instead, depicts the list of revoked certificates containing serial number and revocation date.

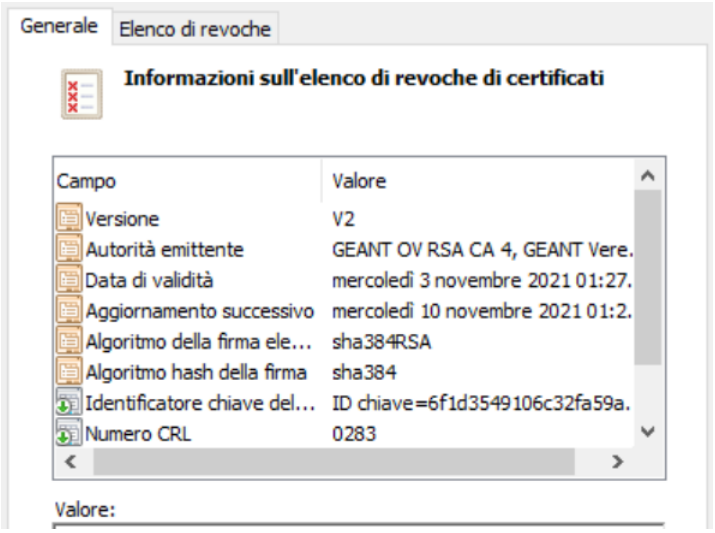


Figure 2.15: Fields of CRL downloaded from url present in CRL distribution points extension of certificate hosted by `www.polito.it`

The negative aspect of retrieving revocation information of certificates through CRL is that browsers must download the entire CRL (containing all revoked certificates by that CA) even if they are interested in the revocation status of one certificate. Since CRLs can assume large dimension (order of Megabyte), downloading CRLs decrease the performance of web browsers in establishing a secure connection. Additionally, they need to be updated periodically and so caching them for a long time is a security issues for the clients.

2.3.2 OCSP

OCSP protocol (standardized in RFC-6960 [15]) was born for resolving drawbacks of revocation checking process through CRLs and allows client to query an authorized and trusted (issued by CA) OCSP server (called OCSP responder) for retrieving the revocation status of a single certificate.

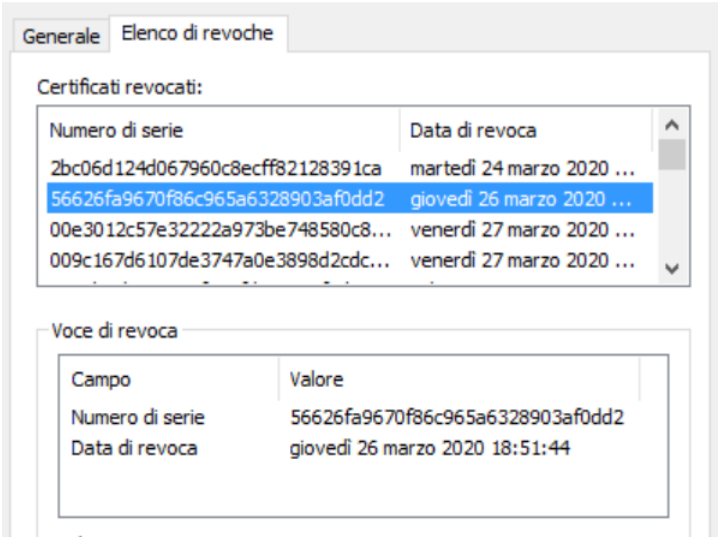


Figure 2.16: List of revoked certificates appearing in the CRL downloaded from url present in CRL distribution points extension of certificate hosted by www.polito.it

When a client needs to retrieve revocation information about a certificate through the OCSP protocol, it builds a HTTP request (indicating the interested certificate's serial number, hash of the issuer's name and public key so that a CA checks whether it has issued the certificate) and sends it to the URL present in the AIA extension. The OCSP responder answers returning a signed OCSP response containing:

- certID: queried certificate's serial number;
- thisUpdate and nextUpdate: since OCSP responses can be cached by clients for a limited period of time, these fields allow to reconstruct the validity period of the OCSP response;
- producedAt: time of OCSP response's generation by the OCSP responder;
- certStatus: certificate status which can be:
  - Good: certificate in good state, it has not been revoked;
  - Revoked: certificate has been revoked;
  - unknown: responder is not able to determinate the status of the certificate.

OCSP protocol resolves CRL problem related to huge dimension of the list, but it requires querying CA for checking the revocation status of certificate every time a client want to establish a secure connection. Additionally, it introduces a potential privacy risks for the users since browsers make request to CAs every time a user try to establish a secure connection with a website and in this way CAs are able to track user's behaviour on internet. Finally, certificate's revocation information depends on the availability and performance of OCSP responders which should provide responses with low latency and large availability.

OCSP stapling OCSP stapling is a standard which let server able to cache OCSP response and send it to clients during TLS handshake as part of *TLS Certificate Status Request* extension. When a client establishes a secure connection through TLS with a web server that support OCSP stapling, it will receives the server's certificate together with the OCSP response attesting the end-entity certificate's validity. At this point clients are able to verify the status of end-entity certificate and being sure the certificate has not been revoked.

Figure 2.17 depicts the OCSP response present in the *TLS Certificate Status Request* extension when try to establish a TLS connection with www.sony.com. The certificate returned by the server is good, the response has been computed on 31 of October and expires on 7 of November.

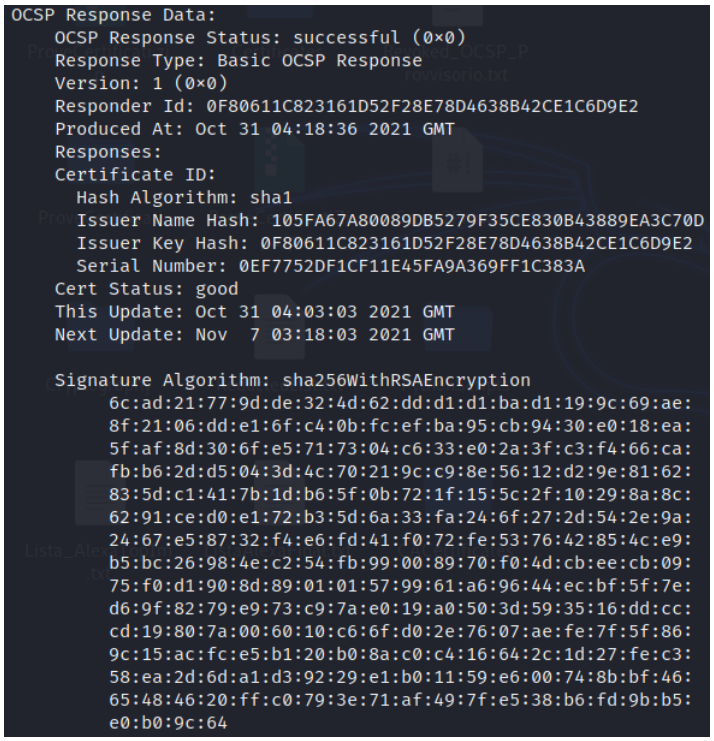


Figure 2.17: OCSP response present in OCSP stapling TLS extension when establish a TLS connection with www.sony.com

OCSP stapling partially resolves latency problem related on generating additional OCSP requests every time a client needs to retrive revocation information about certificates present in a chain, because OCSP stapling includes information only for the leaf certificate. This means that clients may perform OCSP queries for the other certificates in the chain and this will affects client performance on establishing a secure connection. The solution is to include a new extension in TLS standard (proposed in RFC-6961 [16]) which allows servers to forwards multiple OCSP responses to the clients during TLS handshake.

OCSP must stapling OCSP must stapling is a X.509 certificate extension which allows clients to reject a connection with a web server in case it does not provide a valid OCSP response during TLS handshake.

2.4 Certification path validation algorithm

Every time a certificate-usage system (e.g. web browser) establishes a secure connection with a web server, it has to reconstruct a valid certification path rooted by a trusted CA certificate present in a certificate-usage system trusted list. Each web browser has a list of trusted root certificates used for validating certificates appearing in the web server's certificate chains.

The reconstructed certification path may contain certificates belonging to the chain returned by the web server and one of the CA certificate present in browser root stores. Once the path has been generated, certificate-usage system must validate it taking into account the information contained into certificates and in case the built certification path does not respect some restrictions related to the path length, domain name, certificate usage or policy certificate-usage systems must reject the path just reconstructed.

A standard algorithm that certificate-usage systems (e.g web browsers) should follow for validating certification paths is described in RFC-5280 [14]. The validation algorithm requires that certificate-usage systems (e.g web browsers) iterate through all certificates in the chain, starting

from the root certificate, and validate each certificate's information and critical extensions. In case the procedure ends without any security warning, the path is accepted as good; otherwise it is marked as non valid.

Certificate basic information checking

The integrity and validity of each certificates appearing in the constructed path are checked during the basic information checking procedure. Certificates revocation status and correspondences among issuer-subject fields are also handled by basic information checking process.

Certificates integrity For each certificate present in the built path, the relying party must verify the signature over it using public key bound into the issuer certificate. In case the signature of at least one certificate is not valid, certificate integrity cannot be verified and the certificate is rejected.

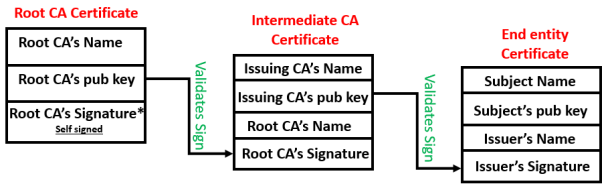


Figure 2.18: Certificates integrity verification in a chain of 3 certificates: relying party must verify signature of each certificate (except for the self-signed) with the public key bound into the previous certificate in the chain

In the example showed in figure 2.18, the relying party must validate the signature over the end-entity certificate with the public key bounded into the intermediate CA certificate and the signature over the intermediate CA certificate with the public key bounded into the root CA certificate.

Certificates validity Relying parties must verify that each certificate appearing in the path has not expired, by checking the current date and time against the certificate validity field. In presence at least of one expired certificate, certificate-usage systems must reject the entire reconstructed path.

Certificates revocation status Since a CA is able to revoke a certificate before its natural expiration date, web browsers must check the revocation status of each certificate appearing in the built certification path by means of CRLs or OCSP. In case at least one certificate belonging to the built path has been revoked, the validation procedure should fail.

Certificates issuer For each certificate of the certification path, relying parties check that a certificate's *issuer* field is equal to the *subject* field of the previous certificate in the path.

Figure 2.19 depicts a chain of 4 certificates: starting from the end-entity certificate, the issuer field must contain the same value of the subject field of the previous certificate and it must be valid for each certificate appearing in the chain, except the root CA where issuer and subject field contain the same value.

Authority Key Identifier (AKI) and Subject Key Identifier (SKI) extensions are helpful for reconstructing the certification path. Another step performed by relying parties during the certification path validation algorithm is the *key identifier chaining*: starting from the end entity certificates and scanning all certificates (except the root ones), the key identifier contained inside the Authority Key Identifier Extension should match the one contained in the Subject Key Identifier of the previous certificate.

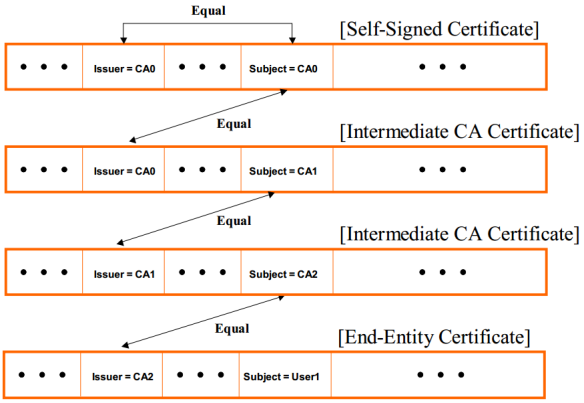


Figure 2.19: Name chaining: starting from end entity certificate and for all certificates in the chain (except for the root CA certificate), issuer field of end entity certificate must be the subject field of the previous one

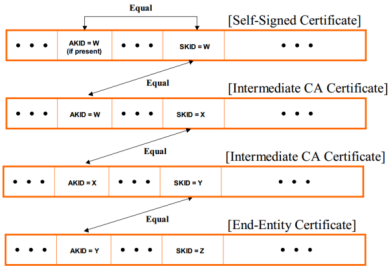


Figure 2.20: Key chaining: starting from end entity certificate and for all certificates in the chain (except for the root CA certificate), Authority Key Identifier field of end entity certificate must be the Subject Key Identifier of the previous one

Figure 2.20 depicts a chain of 4 certificates: starting from the end-entity certificate, the Authority Key Identifier must contain the same value of the Subject Key Identifier field of the previous certificate and it must be valid for each certificate appearing in the chain.

Constraints checking

Certificate extensions allow CAs to impose constraints or restrictions on how certificates they will issue can be validated and handled. Name constraints, policy constraints, basic constraints, key usage and all other critical extensions are the steps performed by web browser in constraint checking phase.

Name constraints During name constraint checking, relying parties check whether certificates in the chain respect limitation (if present) imposed by name constrain extension of previous certificates. Certificates company or organization's domain name can be limited to a specific domain tree expressed by the previous certificate and in case they are not respected, browsers must reject the certificate.

Policy constraints Certificate policy extension lists one or more policy under which certificate has been issued, and in case the extension contains critical policy constraints web browsers must validate them before going ahead.

Basic constraints Basic constraint extension contains the maximum path length a certificate can support. In case the basic constraint extension is set to false, the maximum path length is

set to none; otherwise, in case the extension is marked as critical and it is asserted to true web browsers must verify consistency among the maximum path length and the built certification path.

Key usage Key usage extension is a way for indicating the general use of the public key bound into the certificate. Values of key usage extension may be consistent each other: for example in case the key present in the end-entity certificate can be used for *client authentication* it need to have also *digital signature* indicator set. CA certificates should have *certificate signing* indicator set since they can issue and sign certificates. In presence of some incongruences in the chain, web browsers must reject the built chain.

Critical extensions Finally web browsers end certification path validation process by validating the remaining extensions marked as critical. When a browser reach to the leaf certificate of the path without error, the path is marked as valid and accepted by the browser. In case of error at any level, the path is not accepted as valid and the secure connection with the web server cannot be established.

2.5 Domain Impersonation

Public Key Infrastructure (PKI) and digital certificates allow users to identify the entity they are communicating with by means of certificate chain validation process, but users must be sure the website is what they expect. For this reason users should evaluate domain names, but different kinds of "Domain Impersonation" attacks affect the ability of users in doing so. For example users should be deceived into believing that *apple1.com* is the same of *apple.com* and the presence of a lock icon and the absence of security warnings should trick users in believing they are establishing a secure connection with *apple.com*. Nowadays web browsers perform certificate validation procedure by displaying a lock icon and by loading the requested page when the certificate validation terminates without security warnings, but it does not ensure that the website is what the users expects it to be.

Another critical aspect for the web's security is the ability of attackers in obtaining SSL Domain Validated certificates for web domains they do not own and then use these certificates for creating a malicious copy of websites, affecting users' security. People connecting to these website will not receive any security warning from their browsers since the certificate validation procedure terminates without errors.

The ability of attackers in obtaining a SSL certificate for a web domain they do not own is strictly related with the verification procedures performed by the CA for issuing a domain validated certificate. A CA before issuing a certificate to an entity, for a specific hostname and public key, verifies the claimed entity's identity by means of a challenge sent to the email address specified for the domain or asking to the requestor to include a file into the DNS zone file. The *DNS poisoning* attack allow cybercriminals to pass the validation procedure and to obtain the certificate for the web domain they do not own. As reported in [17], some German researchers have found a way for exploiting this vulnerability and obtaining fraudulent certificates.

Authors of [10] identify a new type of *Domain Impersonation* attack named *target embedding* which embeds a real and unmodified domain inside the actual domain (e.g. apple.com-offers).

2.6 Related works

2.6.1 On the complexity of Public-Key Certificate

Berbecaru et al [2] introduce the certificate validation process starting from the presentation of a famous security incident related to certificate validation. They revise the user and system requirements taking into account multiple constraint such as the computational power of the end-user

client, network connectivity and security policy to be respected. They end by defining a general certificate validation architecture and demonstrating how different certificates management protocol and formats can be adopted in the presented architecture, revealing advantages and drawbacks.

2.6.2 An End-to-End Measurement of Certificate Revocation in the Web's PKI

Liu et al[3] give an overview at certificate revocations in the Web' PKI, discovering that a considerable percentage of served certificates have been revoked and certificate revocation information require an high latency and bandwidth to be acquired. Furthermore, browsers often do not worry to check whether certificates are still valid or not. Liu et al also perform a study on CRLSet infrastructure built into Google Chrome, for spreading revocations, finding that CRLSet only covers 0.35% of all revocations.

Browsers developers often apply a "*Soft-fail*" approach by deciding to trust certificates even when they are not able to load revocation information. It is a positive aspect for usability, but becomes a big issue for PKI security; for this reason a "*Hard-fail*" is advisable, in which browsers do not trust certificates when revocation information are not available by loosing in usability for users. Liu et all suggests to use a "*Hard-fail*" approach which would better inform users of the potential security risks, and may apply useful customer pressure on CAs with unreliable services.

Liu et al study results show that website administrators infrequently enable OCSP stapling, which was developed to address the limitations of CRLs and OCSP. In particular results show that some CAs have not adopted smaller CRLs, enforcing clients to download large CRLs before fully establishing the TLS connection.

Additionally authors performs some tests to study client (browsers) behaviour in certificate validation process: they observe that there are not any browsers in its default configuration correctly checks all revocations and rejects certificates if revocation information is not available. Many browsers do not correctly interpret *unknown* OCSP responses, not all browsers support OCSP stapling and many of them do not understand *revoked* staples. In mobile browsers, instead, there is a complete lack of revocation checking.

CRLSets is a small, pre-populated list of revoked certificates updated and sent to client out-of-bands in order to reduce the cost of checking revocation information at page load time. Authors of [3] investigate their impact on security in matter of revocation finding that it has limited coverage, frequently updates and many experience outages.

2.6.3 Is the Web Ready for OCSP Must-Staple?

Chung et al [4] perform a study to determinate whether all parties involved in PKI (certificate authorities, web server administrators and web browsers) are ready to support OCSP Must Staple.

CAs need to run OCSP responders that are highly available to provide OCSP responses to web servers. Results show that 36.8% of OCSP responders (among the set used for the scope) resulted at least one outage for few hours. Since OCSP responses can be cached and their median validity periods are a week, authors believes that the OCSP responders availability is not an obstacle to the spread of OCSP Must Staple deployments. While offering excellent usability performance and security properties, the number of certificates that supports OCSP Must Staple is very low: only the 0.02% od the certificates analysed by the authors.

Chung et all build a test suite for web browsers and apply it to most popular ones (both desktop and mobile devices). They find that the percentage that correctly support and manage OCSP Must Staple is quite small: only Firefox worry to ensure that stapled OCSP responses are actually included.

Neither Apache and Nginx prefetch and OCSP response, introducing unnecessary latency in terminating the TLS handshake with the client. Additionally, Apache returns expired OCSP

responses from the cache and discards previous, valid OCSP responses when a transient error, in communicating with the OCSP responder, occur.

Chung et al concludes saying that currently web is not ready for OCSP Must Staple.

2.6.4 You are who you appear to be

Roberts et al [10] presents a new classification of impersonation attack named target embedding (it belongs to "Subdomain Spoofing" class of attacks) which does not modify the impersonated domain but uses a subdomain of the actual domain (for example apple.com-signing.id embeds the target domain apple.com while actual domain is com-signing.id. Let's encrypt has released a certificate to this domain on 2018). Authors performs a user study in order to understand whether users are subjected to this kind of attack: results show that users are more vulnerable to target embedding attack than other ones like typosquatting, bitsquatting or combosquatting.

Since URL is the true indicator of a website's identity, users often make their trust decisions based on the page's content, which is easy to replicate. Authors study highlights that if a user falls for a target embedding attack once, they can fall for it several times: it is the most successful means of appearing to be someone a domain is not.

Study conducted by authors reveal target embedding is much more strongly correlated with unsafe domains and it is also able to scale to a much larger set of target domains, and in so doing, is able to identify many more unsafe domains.

By analysing results, Roberts et al say the most targeted domains are relatively unpopular (5 of the top 20 most targeted domains have an Alexa ranking over 500), but many of the most targeted entities represent a clear economic incentive for an attacker that tries to retrieve credentials for bank accounts or cloud storage services.

Many of unpopular TLDs (.ga, .ml, .cf, .tk and .gq), according to Alexa, are among the most used for target embedding together with some of the most popular across all certificates (.com). An attacker choose the TLD to use based on TLD cost and keywords relevant to the target (can be useful to steal passwords confusing users).

Analysing the set of certificates related to target-embedding domain it is possible to note the use of certificates for target embedding is a relatively recent phenomenon: before 2016 there were a small number of such certificates, while with the introduction of Let's Encrypt (which provide free and automated certificate issuance), in 2016, this number increased. Users who obtain certificates for target embedding do not use various CAs, but essentially Let's Encrypt.

On the contrary there is a wide range of providers who host these domains because offer options for free hosting. In this way an attacker is able to acquire, host and secure target embedding domains for free.

Wildcard certificates give the possibility for the owner to perform target embedding on a large number of target; with the introduction of Let's Encrypt, author say, the number of wildcard certificates rise a lot.

Unfortunately there is no one clear fix for target embedding, the only way to mitigate this problem is coordination among multiple players.

2.6.5 Mission Accomplished? HTTPS Security after DigiNotar

Amann et al [6] explore new security features which have been added to TLS, HTTPS and PKI over the past five years such as Certificate Transparency (CT), HTTP Strict Transport Security (HSTS) and HTTP Public Key Pinning (HPKP) headers, Certification Authority Authorization (CAA) and TLS Authentication (TLSA) DNS-based extensions and Signalling Cipher Suite Value (SCSV). In particular they investigate the usage of these technologies (focusing on Certificate Transparency and new TLS and HTTPS extensions), how are used in combination and which protection level is achieved.

To evaluate the use of Certificate Transparency, author extract and validate SCTs from both active scans and passive observations. Active measurement results show SCTs in certificate extensions are the most popular, but SCTs are also commonly present in TLS extensions. Although SCTs via certificate extensions is the most widespread technique, SCTs via TLS extensions are commonly used by popular domains, while SCTs in OCSP staple are rarely present. In addition, results show that most certificates are logged by more than one log operator already and certificates logged in one log only are rarely and largely present in Symantec's Deneb log in which all domains in the issued certificates are truncated to the second-level domain, so excluding subdomains. In this way certificate validation against a Deneb log signature requires the modification of the received certificate truncating all domains contained inside. Another strange situation noted by authors in some certificates is the presence of an extension with an SCT object identifier, without any SCT data.

After checking headers consistency for all domains, authors find only 3.5% of the domains with consistent and HTTP-200 headers support HSTS while only 0.02% support HPKP. Amann et al analyse also HPKP and HSTS attributes:

- Max-Age: domain owners typically set much higher max-ages for HSTS than HPKP;
- IncludeSuDomains: most HSTS domains (56%) use this attribute, while a minor fraction HPKP domain (38%) enable it;
- Preloading Lists: very low usage among general population (both for HSTS and HPKP) while significantly present among top domains;
- Public Key Pinning: an high (86%) percentage of scanned HPKP domains use HPKP correctly.

Authors measures SCSV downgrade prevention support finding most of domains (96%) correctly abort TLS connections appropriating.

Scans also reveal CAA DNS extension has a larger deployment in the Top 1M compared to TLSA one, but more TLSA domains are DNSSEC protected rather than CAA ones.

Finally authors show which HTTPS security extensions protect against specific attack vectors and it highlights most protection mechanism defend against exactly one attack vector (only HPKP and TLSA overlap and protect against MITM attacks). Most of domains deploy only one or two protection mechanisms, and only 2 domains enable all security mechanisms presented by the authors (SCSV, CT, HSTS, HPKP, CAA, TLSA).

Only *google.com*, among Alexa Top 10 domains, deploys TLSA; while TLSA is not used by anyone, SCSV is present in the 90% of Alexa Top 10 domains.

Although presented mechanisms increase the level of security and would have been able to prevent or mitigate the impact of DigiNotar compromise, results suggest they are scarcely used. Technologies (such as Certificate Transparency and SCSV) which are easy to deploy and have small risk to availability are the most used; while those that have either high deployment effort or carry a high risk of misconfiguration have often low deployment. Findings support idea a huge fraction of operators does not care enough about additional security those mechanisms introduce for their sites.

2.6.6 MBS-OCSP: An OCSP based certificate revocation system for wireless environments

The design of X.509 certificate-based secure applications for wireless devices is an open issue because this kinds of devices present some computational, network and storage limitations. One of the critical aspect is the distribution of X.509 certificates revocation status among mobile devices and CPC-OCSP is an adaptation of OCSP protocol which optimize OCSP in wireless environments. Berbecaru et al [7] present MBS-OCSP, an improvement of CPC-OCSP system, based on Merkle hash trees and suitable for wireless environments allowing clients to cache some

received information for future usages. The presented system is flexible because the two endpoints of a communication (client and server) must not agree in advance on any parameter for caching management. The author end by comparing MBS-OCSP with OCSP, CRL and CPC-OCSP in terms of the computational effort and message size.

2.6.7 Tracking adoption of revocation and cryptographic features in X.509 certificates

Authors of [9] perform an interesting measurement study of cryptographic strength and the assumption of revocation mechanisms in the X.509 certificates, by analysing OCSP stapling, RSA public key collisions and the strength of certificate serial numbers. They notice how the usage of these features increased among the 2011-2020. This study is useful for identifying problems and deficiencies in certificate issuance procedures of CA such as: weak serial number, public key collision problem (issuance of the same public key across different certificates for different entities) and lack of revocation. Zulfiqar et al show that adoption of OCSP stapling and OCSP extensions has grown up to 97% and top 6 Cas issued the certificates with a serial number longer than 30. Finally they found 803 public key collision in the dataset.

2.6.8 On the validation of Web X.509 Certificate by TLS interception products

Wazan et al [8] analyse the behaviour of HTTPS interception products (proxies and anti-virus programs) in validating X.509 certificates. They also study how web browsers handle revocation information, focusing on the OCSP stapling mechanism.

Authors performs a series of tests to study the behaviour of some HTTPS intercept products such as Avast Antivirus Free, Kaspersky Total security, AVG Intenet Security, ESET Internet Security, Squid, Charles Web debugging proxy, Mitmproxy and Telerik Fiddler. Wazan et al focus tests on the following X.509 certificate's fields: subject, key usage and certificate status.

Tests helped authors to identify 3 types of different behaviours followed by the HTTPS interception products and for each of them, they define a custom name:

- Full validation (fV): proxies and anti-virus tools handle validation of certificates itself. Kaspersky, Mitm, Squid and Fiddler act in this way;
- Delegated validation (dV): proxies and anti-virus tools delegate certificates' validation to web browsers, while they perform revocation checking. Avast, ESET and AVG act in this way;
- Incorrect validation (iV): proxies and anti-virus tools delegate certificates' validation to web browsers, but they do not perform any kind of revocation checking. Charles proxy behaves in this way.

Tests related to certificate subject name - common name (SCN) field and Subject Alternative Name (SAN) extension show that when the server's identity is null, in 2017 only Squid proxy refuse the connection while in 2019 also Mitm proxy behaves like Squid. The other products that behave following a Full validation approach, and belong to the same category of Squid and Mitm, produce a warning and leave the freedom of choice to the users. When the SAN does not contain the identity of the server and only the SCN field is populated all products accept the certificate other than Squid proxy. Finally in case the SCN is populated with the identity of the server and the SAN contains an entry for the IP address, all products belonging to fV class accept the certificate except Squid proxy which returns a security warning. This happen because Squid proxy set the IP address in a DNS entry inside the SAN extension instead of inserting it in an IP entry.

Wazan et al performs also some tests related to Key Usage and Extended Key usage extension, in order to analyse the behaviour of HTTPS interception products in presence of wrong values

inside the two extensions. Squid is the product which behaves well in handling wrong key usage and extended key usage values, failing only in two tests cases. Other ones, instead, accept the certificates containing wrong values in the majority of tests.

Finally authors focus on the revocation checking process, introducing the criticism related to this important step and the protocol can be used for retrieving revocation information. They implement a Java program for detecting how many web servers belonging to Alexa Top 1 Million sites support OCSP stapling. Results show that OCSP stapling was supported by the 19% of web servers in 2017 and by the 27% in 2019; only 58 certificates contains must staple extension in 2017 and 0 in the 2019; only 1 server in in 2018 and 2019 support. Two additionally experiments are conducted by the authors: the first for checking the OCSP stapling support in web browsers and the secondo for studying the reaction of HTTPS interception product under different conditions related to the availability of revocation information. Results show that all tested browsers support OCSP stapling but only Mozilla supports OCSP Must Stapling. Only 3 HTTPS interception products support OCSP (Kaspersky, fiddler and ESET) and the same for CRL checking (AVG, Avast and ESET).

Wazan et al conclude saying that HTTPS interception products perform a bad certificate validation procedure even worse of the one performed by web browsers and they say it could be related to several aspects such as the complexity and vagueness of existing standards.

Chapter 3

Certificate Transparency

Nowadays web clients (e.g browsers) are able to detect dangerous and harmful sites advertised with false SSL certificates, but they are not in charge of detecting dangerous sites advertised by certificates have been issued by a compromised CA. This is a critical aspect for PKI security, because browsers do not notice anything strange and dangerous for clients as CA seems to be in good state. The problem is that there is not an easy and strength way for monitoring and auditing SSL certificates in real time, so that these kinds of certificates can be detected as soon as possible; on the contrary, they are detected after weeks or months from their issuance and during this amount of time clients have been exposed to threats. In the last years, the amount of misissued certificates has grown dramatically, in particularly they have been used for installing malicious software, tracking users positions or stealing users information.

DigiNotar incident in 2011 was a critical event in web history, where attackers took control of famous Dutch CA being able to issue rogue certificates. DigiNotar was present as trusted CA in most browsers root stores and has exposed clients to enormously risks until it has been removed from the browser trusted lists. Since then, several proposals have been presented to increase web PKI's strength such as Certificate Transparency (CT).

Certificate Transparency (CT) is an open, global and monitoring system based on append-only public logs that collect certificates issued by CAs; it gives the possibility for monitoring each new entry and offers to domain owners a way for detecting fraudulent certificates issuance. This systems points to make CAs unable of issuing SSL certificates for a specific domain, unless it becomes visible for the domain owner. This increase the security of users in web because CT ecosystem protects them from being duped by rogue certificates associated to malicious websites.

Web clients (e.g. browsers) should only accept certificates publicly logged, that are monitored and checked by domain owners which are able to detect whether some CAs have issued rogues PKCs associated to their domain.

Each log is composed of several certificate chains, each one rooted by a known CA certificate. CAs influence CT effectiveness because they may add an entry in one or more logs every time they issue a new certificate.

Actors involved in CT project are:

- Submitter
- Loggers
- Monitors
- Auditors

CT submitters and monitors Submitters are those that submit certificates (or partially completed certificates) to a log server and receive a SCT as response. Monitors, instead, are

public or private services that looks for misbehaving or suspicious certificates. They also ensure that all logged certificates are visible in the log by periodically asking for new entries. In this way, monitors have a sort of backup of the monitored log, which can be used as backup read-only log for other monitors and auditors, in case log itself goes down for a long period of time.

CT auditors Auditors roles can be summarized as follow:

- Verify logs integrity: logs integrity is provided by log proof, which is a signed cryptographic hash of log used to say they are in good state. Auditors periodically asks for log proof and verify them by checking that new entries have been correctly added to old ones and no one has manipulated the log.
- Verify the presence of a particular certificate in a log asking an audit proof: since the CT system requires that all TLS certificates appears at least in a log, this functionality is useful to detect whether a TLS certificate is present in a log during TLS handshake and in case it is not present, TLS client may refuse the connection with the website having the suspected certificate.

Although log proof is used, by monitors and auditors, to verify current version of a log is consistent with the previous ones, monitors and auditors different log views must be consistent each other. In order to achieve this, they exchange information about logs through a gossip protocol.

CT log servers Logs play a crucial role in CT ecosystem because they collect secure logs of TLS certificates, having the following features:

- Append-only: the only operation supported by logs is the insertion. Each TLS certificate can be only added to logs, without possibility for editing or deletion;
- Cryptographically protected: logs are organized in Markle Hash Tree for efficient auditing and to prevent tempering and misbehaviour;
- Publicly auditable: everyone can query a log for checking its correct behaviour or verify that a TLS certificate has been correctly submitted to the log.

3.1 Signed Certificate Timestamp (SCT)

Anyone is able to submit certificates to a log server, although an high percentage of certificates will be submitted by CAs. Every time someone submits a valid certificates to a log, loggers answer with a kind of promise named Signed Certificate Time-Stamp (SCT): the certificate will be logged until a certain amount of time, indicated as Maximum Merge Delay (MMD). SCT will be part of the X.509 certificate for its lifetime and it must be delivered by web servers during TLS handshake and it can be provided in three way: as part of X.509v3 extension, TLS extension or OCSP stapling.

SCT via X.509v3 extension Figure 3.1 depicts SCT delivered via X.509v3 extension:

1. CA, before issuing the real certificate, submits a pre certificate to the log server. This pre-certificates contains a critical poison extension to be sure it will not be treated as a standard certificate by TLS clients;
2. Log server returns a SCT;
3. CA attaches the SCT to the already issued pre certificate as a X.509v3 extension, removes the poison critical extension, signs the certificate and sends it to the server operator.

This solution does not require any changes in servers, as server operators manage SSL certificates as before while CAs may change a bit the issuing certificates procedure.

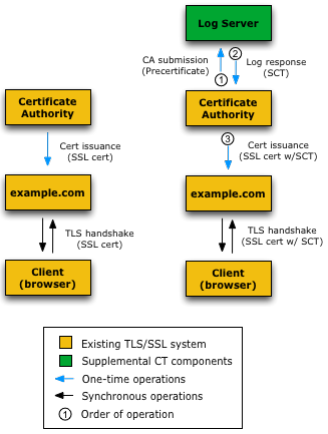


Figure 3.1: SCT via X.509v3 extension
https://sites.google.com/site/certificatetransparency/how-ct-works

SCT via TLS extension Figure 3.2 depicts SCT delivered via a TLS extension:

1. CA issues the certificate (without SCT extension) and sends it to the server operator;
2. The server operator submits the certificate to the log server;
3. Log server sends the SCT to the server operator;
4. The server uses TLS extension signed_certificate_timestamp to deliver the SCT to the client during the TLS handshake.

This solution does not require changing in CAs issuing certificates procedure, while it requires a change in server to populate the TLS extension. In this case the website manager takes care of inserting the certificate in the log.

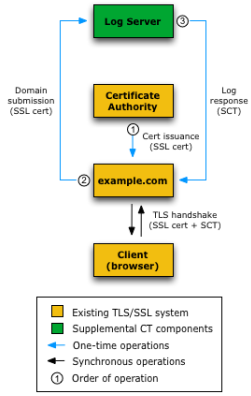


Figure 3.2: SCT via TLS extension
https://sites.google.com/site/certificatetransparency/how-ct-works

SCT via OCSP stapling Figure 3.3 depicts SCT delivered via OCSP stapling:

1. CA issues the certificate (without SCT) and simultaneously sends it to the log server and server operator;
2. Log server responses with the SCT (addressed to the Certification Authority);
3. Server operator queries the CA through a OCSP request;

4. CA returns the OCSP response containing the SCT, which can be inserted by the server in an OCSP extension during TLS handshake.

This solution requires changing on server for making OCSP stapling; CAs are responsible for SCT and they do not delay certificates issuance since SCTs can be retrieved asynchronously.

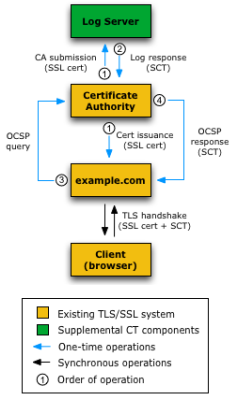


Figure 3.3: SCT via OCSP stapling
https://sites.google.com/site/certificatetransparency/how-ct-works

3.1.1 SCT structure

SCT structure is defined in RFC-6962 [18] and depicted in figure 3.4. As suggested by the figure, an SCT contains:

- sct_version: version of SCT protocol (normally v1);
- id: log's public key SHA-256 hash;
- timestamp: current NTP time in milliseconds;
- extensions: future extensions would be added to this version of protocol;
- signature: cryptographic signature (performed with log's private key) over a structure (referring as signature input) containing: sct_version, signature_type (always equal to "certificate_timestamp"), timestamp, signed_entry and extensions.
- entry_type: depends on the scenario in which SCT appears. Possible values are:
  - precert_entry in presence of a pre certificate;
  - x509_entry in presence of a normal X.509 certificate.
- signed_entry: structure whose content change according to entry_type value, as showed in figure 3.4.

As figure 3.4 depicts, "signed_entry" is different for a precertificate and a standard SSL certificate:

- Pre-certificate: "signed_entry" is a structure composed of issuer key hash and TBS part of received pre-certificate without poison critical extension. To be sure the issuer has logged the pre certificate will also issue the final certificate, the signature contained in signed_entry will be performed also over the pre-certificate issuer public key.
- Standard certificate: "signed_entry" is the received certificate.

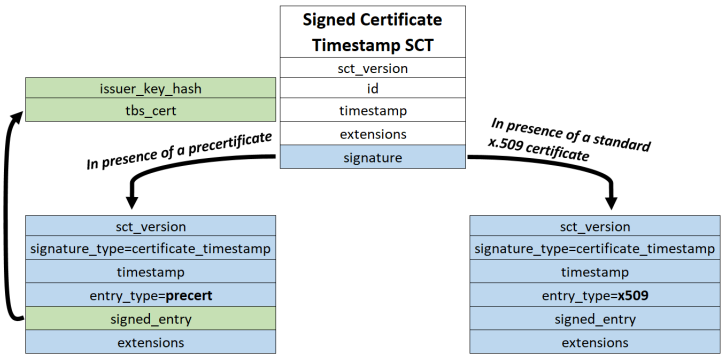


Figure 3.4: SCT structure

```
kali@kali:~/Desktop/Certificates$ openssl x509 -in www-polito-it.pem -text -noout
```

Figure 3.5: Command run to inspect X.509 certificate content

Through openssl library, I have inspected the content of SSL certificate advertised by `www.polito.it`:

The certificate supports the "CT Precertificate SCT" extension and it lists 2 SCTs as depicted in figure 3.6. Each SCT is composed of:

- Version: 0x0 indicates version 1 of the protocol;
- LOG ID: SHA-256 hash of Log's public key;
- Timestamp: time when the log received the pre-certificate from the CA;
- Extensions: in both cases there are not extensions;
- Signature: log's signature over SCT precedes by the signature algorithm (ecdsa with SHA-256 in both cases).

I looked for a web server which delivers SCTs also via TLS extension. I found that "ritter.vg" delivers SCTs both via X.509 extension and TLS extension. In order to retrieve the one delivered via TLS extension number 18, I run the command in figure 3.7: the command establishes a TLS connection with the web server and explicitly requires the TLS extension number 18.

I have analysed the content of "Server Hello" packet with Wireshark as showed in figure 3.8. The "signed_certificate_timestamp" extension contains several entries and only 4 of these logs are known by web server.

3.1.2 Validation of a real SCT

I validate a real SCT embedded in the certificate hosted by `ritter.vg`, by checking whether it has been correctly appended into the public log has made the promise.

Testbed configuration Firstly I establish a connection with `ritter.vg` and I capture the packet exchanged with Wireshark. I explore the packet containing the server's certificate and I select one of the *Signed Certificate Timestamp* contained in the end-entity certificate. The SCT I decide to verify is the one depicted in figure 3.9 which has been issued by *Cloudfare Nimbus 2021*.

I also download the certificate hosted by `ritter.vg` and I inspect the content with OpenSSL as depicted in 3.10: the first SCT appearing in the picture is the one I select for the verification.

```
CT Precertificate SCTs:
    Signed Certificate Timestamp:
        Version   : v1 (0x0)
        Log ID    : 46:A5:55:EB:75:FA:91:20:30:B5:A2:89:69:F4:F3:7D:
                    11:2C:41:74:BE:FD:49:B8:85:AB:F2:FC:70:FE:6D:47
        Timestamp : Feb 19 10:09:11.355 2021 GMT
        Extensions: none
        Signature : ecdsa-with-SHA256
                    30:44:02:20:43:72:FD:5A:C9:AA:02:5D:98:37:C5:5E:
                    E4:10:CB:62:AF:D7:12:49:B9:8F:CB:12:90:5B:8E:1A:
                    47:50:44:05:02:20:70:E3:2D:F2:C5:CB:FB:2D:15:B8:
                    6E:EF:18:97:84:0A:82:67:DF:81:7E:0D:9D:31:AA:75:
                    5B:B3:CC:22:F0:41
    Signed Certificate Timestamp:
        Version   : v1 (0x0)
        Log ID    : DF:A5:5E:AB:68:82:4F:1F:6C:AD:EE:B8:5F:4E:3E:5A:
                    EA:CD:A2:12:A4:6A:5E:8E:3B:12:C0:20:44:5C:2A:73
        Timestamp : Feb 19 10:09:11.679 2021 GMT
        Extensions: none
        Signature : ecdsa-with-SHA256
                    30:45:02:20:05:58:55:F1:11:13:D7:A6:42:40:F6:DD:
                    A3:8F:52:1C:BC:D6:87:04:0B:39:84:9A:92:5B:E1:DC:
                    25:E9:49:4A:02:21:00:D4:CF:24:06:47:E1:CD:69:6F:
                    B0:C9:75:16:5A:51:81:11:4B:4C:44:AB:B9:7D:2F:3A:
                    34:99:48:39:5F:0E:81
```

Figure 3.6: Content of SCT extension in X509 certificates

```
kali@kali:~/Desktop/Certificates/ProvaLettura$ openssl s_client -serverinfo 18 -connect ritter.vg:443 -servername ritter.vg
CONNECTED(00000003)
---BEGIN SERVERINFO FOR EXTENSION 18---
ABIFLAUqAHYAAwGd8/2FppqOvR+sxtqbpz5Gl3T+d/V5/FoIuDKMHWsAAAFhr+PW
ogAABAMARzBFAiEAvZWB69eJqg0rS+sASakkaC6h5DACikEke7zrp1Y1NEYCIAcD
MhwMoL+Ji+p5hCJvhobMzPXPJ3j/BYMM8kTN/mlBAHcApLkJkLQYWBSHuxOizGdw
Cjw1mAT5G9+443fNDsgN3BAAAAFyBm+MHwAABAMASDBGAiEAg0nTE7fjOvMtSCoK
zrf+i+ziS5d8cDTRuEIwdr7r5TcCIQD9+w1AqO+/DsO4flWd5koxOYvzs4YPs8BQ
C6M8IitdIQB3ANt0r+7LKeyx/so+cW0s5bmquzb3hHGDx12dTze2H79kAAABZVPV
w/YAAAQDAEgwRgIhAJsqfgAHzM2QqqXHiJm4gYTAmildfB4×6yJ2vDgkl+JTAiEA
sWJvmvEAWU1xPblKnQ+7eD2tXmRPKiCSn1es36sFEaYAdQBep3P531bA57U2SH3Q
SeAyepGaDIShEhKEGHWWgXFFWAAAAXSOP1moAAAEAwBGMEQCICSF+GCU/2BH6cWK
AN/iAOffIjJs2GUR08qSMYBYoKL9AiAdBsE08ad+/nvyw7w+NsX8GiiSWxR/f/2x
zmvWnN5IqQB2ACk8UZZUyDlluqpQ/FgH1Ldvv1h6KXLcpMMM9OVFR/R4AAABcNBa
fqoAAAQDAEcwRQIhAP0o+clgsHtpG+huGlQUpFPW16n5FTeens5r1Hr4GjMUAiAO
lV8NjSOz0ZIX48alW6PaTcHenhblJp8if3WFflAD8gB2AESUZS6w7s6vxEAH2Kj+
KMDa5oK+2MsxtT/TM5a1toGoAAABeH5uktEAAAQDAEcwRQIhALkhW0qaqmiHcd7f
qrsyv6W6n+M0RoYLq1Q4vUNqIP5oAiAalVWQmGesRa15OlZA/q20oo2M/Bnzq63e
H9ADTjIv6wB2AO5Lvbd1zmC64UJpH6vhnmajD35fsHLYgwDEe4l6qP3LAAABcNBa
fA4AAAQDAEcwRQIhALJ/n0vEBUwUeQU3gcGCINqsj9FoZw3/pq8gA7vt/GzpAiA9
nebsZn2teQ/IrRojPNneKgCBRao0ahXumrgIgyLYlgB3AG9Tdqwx8DEZ2JkApFEV
/3cVHBHZAsEAKQaNsgiaN9kTAAABeH5uoawAAAQDAEgwRgIhAMOPeSmi9sCWKU9L
UloU5fIy4xDYDIffoeRoxLM92GC4AiEApxWRiwKGu6IgC440kB8EVvwytr+57TNV
xvmg9iK6XDkAdgB0ftqDMa0zEJEhnM4lT0Jwwr/9XkIgCMY3NXnmEHvMVgAAAWzZ
0iAYAAAEAwBHMEUCIQCmh/kSR+k8+/XIcPtjatRDSCOsPtzxG7VD++DztCzhVwIg
fir5RPKT6o1c9kidCY5J46lQe/Q8H5NUGb46GxGR6DgAdgBVgdTCFpA2AUrqC5tX
PFPwwOQ4eHAlCBcvo6odBxPTDAAAAXh+bp7FAAAEAwBHMEUCIQDjyxHuYrzYT0En
iksp09B3UoYLLZh5qyqDbb9ZQdBzgQIgOKUlV3wYt6mZJNbbXNLADO40zPcoVxxl
RfpdlASaI2YAdgCHdb/nWXz4jEOZX73zbv9WjUdWNv9KtWDBtOr/XqCDDwAAAXDQ
WpBPAAAEAwBHMEUCIHTH43CrPyxBPzzBEvc2UG62wD9NzeofhVkiWDCtwCkoAiEA
/mv6xQT5n3SN991AT4RAwF1xdCi78rVcTDjyPegJUN8=
---END SERVERINFO FOR EXTENSION 18---
```

Figure 3.7: Command run for retrieving SCT via TLS extension

Log ID The LogID value appearing in the SCT allows to retrieve the CT log has signed the timestamp. The web site https://www.gstatic.com/ct/log_list/v2/log_list.json maintains the list of CT logs compliant with Chrome's CT policy. Algorithm identifier that appears inside the chosen SCT is 0403 which stands for ECDSA with sha-256. I save the key of *Cloudfare Nimbus 2021* in a file named Nimbus.key and I compute the Sha-256 digest of the public key in binary format as indicated by RFC-6962 [18].

The computed Log Id, showed in figure 3.11, is equal to the one present in the SCT entry showed in 3.9 and in figure 3.10. This helps me on understanding that the SCT has been signed by *Cloudfare Nimbus 2021*.

Inclusion checking In order to be sure whether the certificate hosted by ritter.vg has been correctly issued in a public log, I visit the web site https://transparencyreport.google.com/https/certificates?hl=en where site owners can verify for incorrect issuances of certificates referencing their domain.

I use this web site for retrieving information about the certificates issued for the domain ritter.vg, as depicted in figure 3.12, and I look for the certificate with the same serial of the one inspected with Wireshark and OpenSSL. The certificate I am looking for has the following

Figure 3.8: Content of server hello packet for web server www.ritter.vg

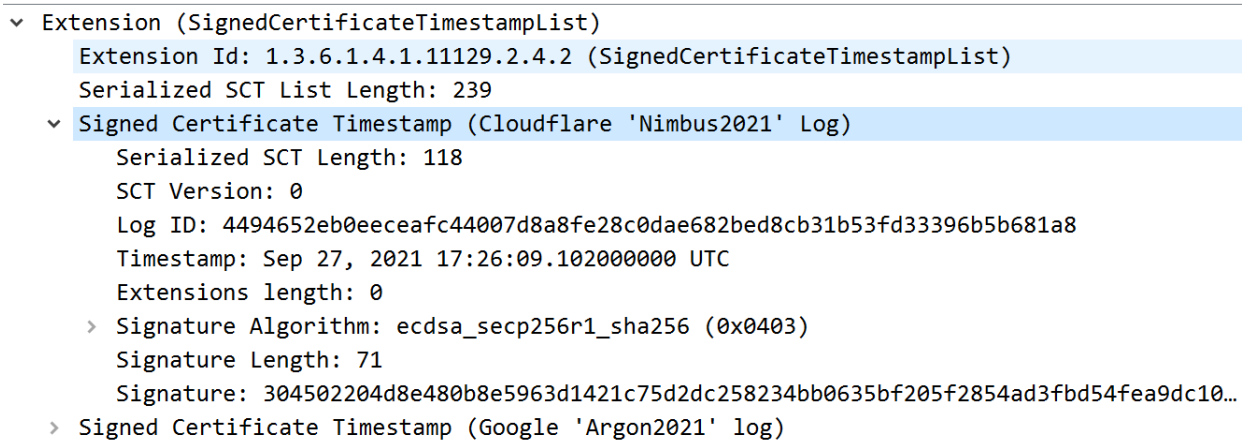


Figure 3.9: One of the SCTs present in Signed Certificate Timestamp extension of certificate hosted by ritter.vg

serial number: 04:80:98:96:e3:92:cc:9c:62:b6:cc:cd:3f:56:7b:05:45:36 and by inspecting the list one entry at time I found that the second certificate of the list is the one I was looking for.

I inspect the content of the certificate and, as figure 3.13 suggests, the certificate has been correctly logged into *Cloudfare Nimbus 2021* log and it is placed in position number 397895328.

Conclusion I verified that the certificate hosted by *ritter.vg* has been correctly appended into the *Cloudfare Nimbus 2021* log and so the promise present as form of SCT inside the X.509 certificate has been respected by the public log.

3.2 Log proofs

Certificate Transparency uses Markle hash trees to make public auditing of certificates and logs as smooth as possible. Merkle hash tree presents a binary tree structure of hashed nodes and leaves, as depicted in figure 3.14. Each leaf represents one certificate hash which have been inserted into the log, while nodes are defined as the hash of a leaves pair. The tree hash root, in which nodes flows together, is named Merkle tree hash and it is the result of hash operation among first level nodes. When monitors request current situation of a log, logger returns all log entries together with the Signed Tree Head (STH) which is the Merkle tree hash signed by the logger itself (Steps E-F figure 3.17).

```
CT Precertificate SCTs:
    Signed Certificate Timestamp:
        Version   : v1 (0×0)
        Log ID    : 44:94:65:2E:B0:EE:CE:AF:C4:40:07:D8:A8:FE:28:C0:
                    DA:E6:82:BE:D8:CB:31:B5:3F:D3:33:96:B5:B6:81:A8
        Timestamp : Sep 27 17:26:09.102 2021 GMT
        Extensions: none
        Signature : ecdsa-with-SHA256
                    30:45:02:20:4D:8E:48:0B:8E:59:63:D1:42:1C:75:D2:
                    DC:25:82:34:BB:06:35:BF:20:5F:28:54:AD:3F:BD:54:
                    FE:A9:DC:10:02:21:00:9D:03:84:47:07:BE:24:62:06:
                    EE:83:05:2B:5C:84:6B:09:C2:84:60:89:E3:05:0E:C3:
                    06:1F:A1:F8:EE:0A:DF
    Signed Certificate Timestamp:
        Version   : v1 (0×0)
        Log ID    : F6:5C:94:2F:D1:77:30:22:14:54:18:08:30:94:56:8E:
                    E3:4D:13:19:33:BF:DF:0C:2F:20:0B:CC:4E:F1:64:E3
        Timestamp : Sep 27 17:26:09.049 2021 GMT
        Extensions: none
        Signature : ecdsa-with-SHA256
                    30:44:02:20:61:BE:0B:7D:30:5E:87:2F:01:49:D3:F4:
                    37:70:D4:02:5E:D1:2D:42:1E:AC:81:24:0B:6D:A4:FB:
                    AA:47:B8:63:02:20:38:D0:31:33:45:85:17:1C:58:0D:
                    29:70:83:27:F1:EA:46:BE:F2:A8:D8:BF:50:F3:B1:00:
                    5E:F3:F0:7C:A3:C2
```

Figure 3.10: List of SCTs embedded in the certificate hosted by ritter.vg

```
kali@kali:~/Desktop/Certificates/SCT_Verification/Pre_Cert$ openssl ec -in Nimbus.key -outform der -pubin 2>/de
 -sha256 -binary | hd
00000000  44 94 65 2e b0 ee ce af  c4 40 07 d8 a8 fe 28 c0  |D.e......@....(.|
00000010  da e6 82 be d8 cb 31 b5  3f d3 33 96 b5 b6 81 a8  |......1.?.3.....|
00000020
```

Figure 3.11: Computation of Log ID for Cloudfare Nimbus 2021 Log

Regularly loggers add received new certificates to logs, by computing a new and different Merkle tree hash with the last appended certificates. Then it will be merged with the old Merkle tree hash to generate a final version of the tree. Finally, the built tree will be signed to create a new signed tree head. It is a recursive process, that will generate an ever-growing Merkle tree of all certificates added to the log.

Logs, thanks to Merkle hash tree structure, can prove in an easy and quick way two important aspects:

- Presence of a target certificate in the log, after an append operation, providing the so called Merkle audit proof;
- Consistency among appended certificates in the log providing the so called Merkle consistency proof.

3.2.1 Merkle audit proofs

Since CT project expects that TLS client reject connection with web servers whose certificate is not present in at least one log, Merkle audit proof helps on verifying whether a certain certificate appears in a log. It is the list of missing node hashes necessary for the computation of the nodes among the target leaf certificate and the root.

To verify whether certificate C4 appears in the log showed in figure 3.16, the Merkle audit proof consist of the following node hashes: N1, H3 and M2. Starting from these nodes, auditors

Subject	Issuer	# DNS names	Valid from	Valid to	# CT logs	
ritter.vg	R3	1	May 29, 2021	Aug 27, 2021	2	See details
ritter.vg	R3	1	Sep 27, 2021	Dec 26, 2021	2	See details
ritter.vg	R3	1	Mar 29, 2021	Jun 27, 2021	2	See details
ritter.vg	R3	1	Jan 25, 2021	Apr 25, 2021	2	See details
ritter.vg	R3	1	Sep 27, 2021	Dec 26, 2021	2	See details
ritter.vg	R3	1	Mar 29, 2021	Jun 27, 2021	5	See details
ritter.vg	R3	1	Jul 28, 2021	Oct 26, 2021	2	See details
ritter.vg	R3	1	Jul 28, 2021	Oct 26, 2021	2	See details
ritter.vg	R3	1	Jan 25, 2021	Apr 25, 2021	5	See details
ritter.vg	R3	1	May 29, 2021	Aug 27, 2021	2	See details

Figure 3.12: List of Certificates issued for the domain `ritter.vg` in the last year

Dettagli del certificato

Oggetto: CN=ritter.vg
Numero di serie: 4:80:98:96:E3:92:CC:9C:62:B6:CC:CD:3F:56:7B:05:45:36
Emittente: C=US, O=Let's Encrypt, CN=R3
Validità: 27 set 2021 — 26 dic 2021

Google invita tutte le autorità di certificazione (CA) a scrivere i certificati emessi in log a prova di manomissione, di tipo append-only e pubblicamente verificabili.

Di seguito è riportato un elenco con i log di Certificate Transparency in cui è stato registrato il certificato e la posizione del certificato nel log.

Log di Certificate Transparency

Log	Indice
cloudflare_nimbus2021	397895328
google_argon2021	1323377193

Figure 3.13: Detail of the last issued for the domain `ritter.vg`

are able to compute the Merkle tree hash (hash of the root node) and compare it with the one of the log. In case the two values are consistent, the target certificate has been inserted into the log and the TLS client can accept the connection with web server; otherwise, TLS client must reject the connection.

3.2.2 Merkle consistency proofs

Merkle consistency proof allow to verify two versions of a log are consistent each other: old log version entries must be present in the new version of the log in the same order as before, being careful they precede the new entries. Consistency proof ensure log has not been altered: there are not back-dated certificates added into the log, the ones already present have not been modified and the log itself has not been manipulated.

Merkle tree hash version in figure 3.15 is consistent with version in figure 3.16 because:

- Old version of Merkle tree hash is a subset of the new one;
- New version of Merkle tree hash is the result of concatenation among the Merkle tree hash old version and node hashes of new certificates appended in the log.

Merkle consistency proof represent the minimum set of nodes needed for computing the process explained before.

For the explained scenario, the Merkle consistency proof is composed of node M1, N3 and N4 because they are the essential nodes needed for verifying the presence of old tree structure in the new one and for checking correctly concatenation with new node hashes.

Monitors and auditors take advantage of consistency proof to verify logs are behaving correctly. Since a monitor have a copy of certificates present in a log, to verify log's consistency, it is able to compute consistency proof by itself, and compare it with the one computed by the log.

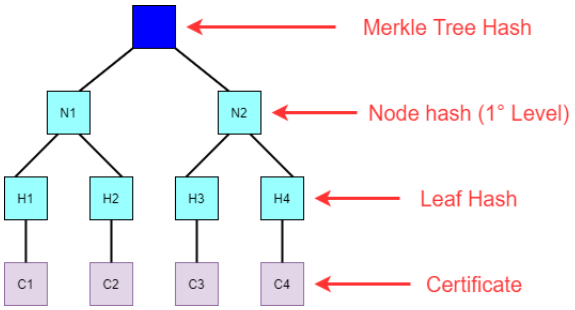


Figure 3.14: Example of Merkle Tree Hash structure

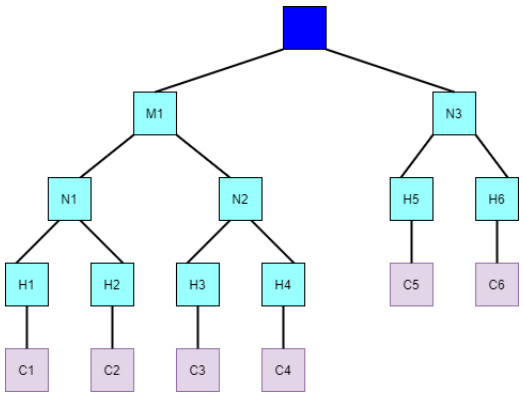


Figure 3.15: Merkle Tree Hash old version

3.3 Interaction among CT entities

Figure 3.17 summarizes interaction among parties in a CT ecosystem:

- Step A: Submitter communicate a new entry to log server;
- Step B: Logger return a signed certificate timestamp (SCT) to submitter;
- Step C: During communication on Internet, web clients must establish a secure connection with web server through TLS and during TLS handshake will retrieve both web server certificate and SCT, that will be forwarded to a CT auditor for verifying its presence in logs. Auditor contacts logger asking for a Merkle audit proof related to the SCT received by the client;
- Step D: Logger sends Merkle audit proof to auditor;
- Step E: Monitor requests a full log to the logger;
- Step F: Logger sends the requested log together with its STH (Signed Tree Hashes);
- Step G: Monitor asks a consistency proof to logger;
- Step H: Logger sends the consistency proof.

3.4 Possible CT system configuration

Since CT project does not require a standard configuration for monitors and auditors inside the SSL ecosystem, there are some configuration more popular than other such as the one depicted in figure 3.17. In this solution a CA is responsible for running a monitor while a browser is responsible for running an auditor:

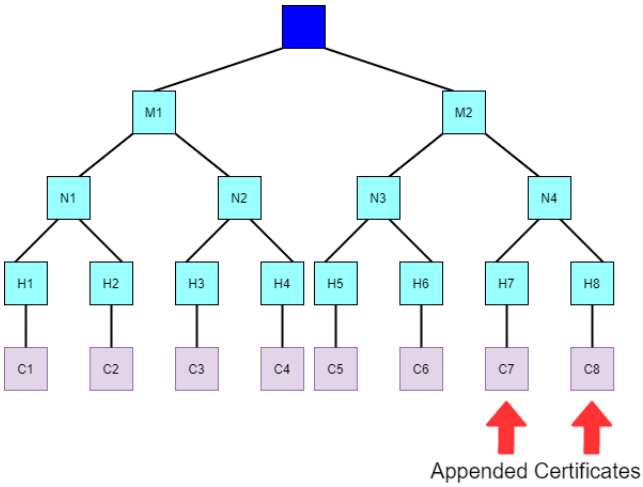


Figure 3.16: Merkle Tree Hash newly version

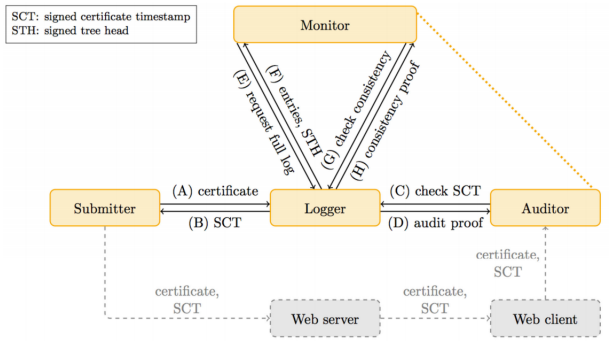


Figure 3.17: Interaction among entities in CT
https://www.douglas.stebila.ca/research/papers/ESORICS-DGHS16/

1. CA receive the SCT from a log and insert it into the appropriate X.509v3 extension of SSL certificate. Monitor run by a CA watch logs for suspected certificates and verify the availability and visibility of all logged certificates;

2. CA issues the certificate (embedded with SCT) to the server operator and web server do not have to change the way they manage SSL certificates;

3. Client requires a TLS connection with the web server and during TLS handshake web server sends to client the SSL certificate and the certificate's SCT. At this point client must validate not only the certificate and its chain but also the log's signature on the SCT to verify SCT has been issued by a valid log and that it has been issued for that specific certificate. Auditors verify logs behaviour and whether current certificate has been already logged. If something goes wrong, TLS client must refuse the connection such as when the SCT timestamp is in the future.

4. Monitor and auditor exchange information about the state of logs useful for detecting wrong log behaviours.

Chapter 4

Analysis of a X.509 certificates dataset

The number of issued certificates grown exponentially in the lasts years and the birth of Let's Encrypt [11] greatly influenced domain owners in choosing the CA that would certify their identity, since Let's Encrypt offer this procedure for free.

In order to depict the current situation of X.509 certificates in the web PKI and make some considerations, I have collected certificates advertised by AlexaTop1M sites updated at 26th of August. I will inspect more than 400.000 certificates and for each one I will check: issuing CA, expiration date and presence of most common extension such as Basic Constraint, AIA, AKI, and SKI.

For this scope I download the AlexaTop1M list updated at 26th of August and I created two scripts:

1. The first script tries to establish a TLS connection with the site passed as argument and then saves the entire certificate chain. In case of failure, the script goes ahead with the next entry. For this purpose I used the openssl s_client command which establish a TLS connection with the host passed as argument;
2. The second script has been created for scrolling the downloaded list and, once per time, passing the read web server to the first script.

This procedure took me 21 days and at the end I have collected 442.331 certificates over a list of 645.332 sites. The number of certificates is less rather then AlexaTop1M list entries because more servers can advertise the same certificate. Additionally there have been many connection errors with several servers which also has influenced the collection of certificates. The set up dataset will be analysed in order to make statistics which will help me to understand which extensions certificates are embedded with, which are the most popular CAs, how many certificates are expired and revoked.

4.1 X.509 fields analysis

Basic Constraint Extension As first thing I have divided the set of certificates into: leaf certificates and CA certificates. Certificates which belong to the first class are normal end entity certificates while ones which belong to the second class are CA certificates with Basic Constraint extension set to true. In order to compute this division I wrote a python script that checks for each certificate the value of basic constraint and writes into a file ones for which the extension is set to true. 442.331 certificates have been processed and among these:

- 1127 certificates (0,25%) have CA value of Basic Constraint extension set to true and they have been written into a file;

- 1799 certificates (0,41%) do not have Basic Constrain extension and raise the 'Extension not found' exception;
- 1 certificate raise a Value Error exception, because path_length value of Basic Constraint extension is not set to None while CA value is false;
- 439.404 (99,34%) certificates have CA value of Basic Constraint extension set to false.

I moved the 1127 CA certificates into another folder, with a simple script that read the name of certificate from file (once per time) and execute the shell move command.

Pictures 4.1 and 4.2 show how extensions are marked separately for leaf set and CA certificates set:

- Leaf set present 407.169 certificates (92,66%) with Basic Constraint extension marked as critical while 32235 (7,34%) marked it as non critical;
- CA set, instead, contains 654 (58,03%) certificates with a critical Basic Extension and 473 (41,97%) with a non critical one.

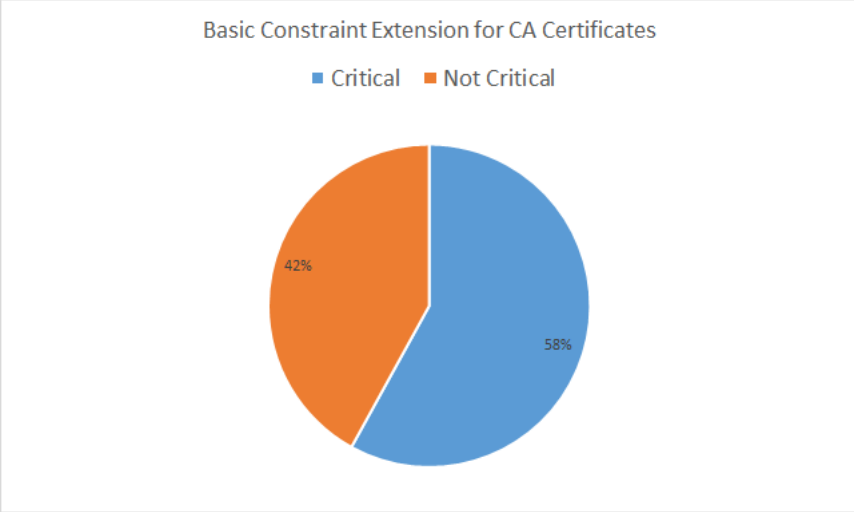


Figure 4.1: How basic constraint extensions have been marked in CA certificates

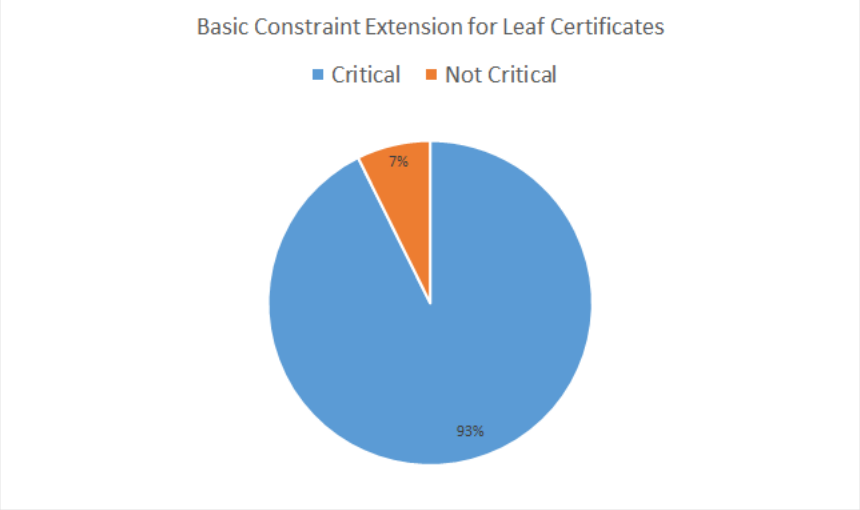


Figure 4.2: How basic constraint extensions have been marked in Leaf Set

Certificate Issuer Let's encrypt is a CA authority that provides X.509 certificates on charge and nowadays is used by the majority of websites (approximately 265 million).

In order to detect how many certificates belonging to the leaf set have been issued by Let's Encrypt, I wrote a python script that looking at the issuer of each certificate increment the correspondent entry in a key-value dictionary created for this purpose. Results in figure 4.3 shows that more than 55% of certificates belonging to the Leaf set have been issued by Let's Encrypt Organization.

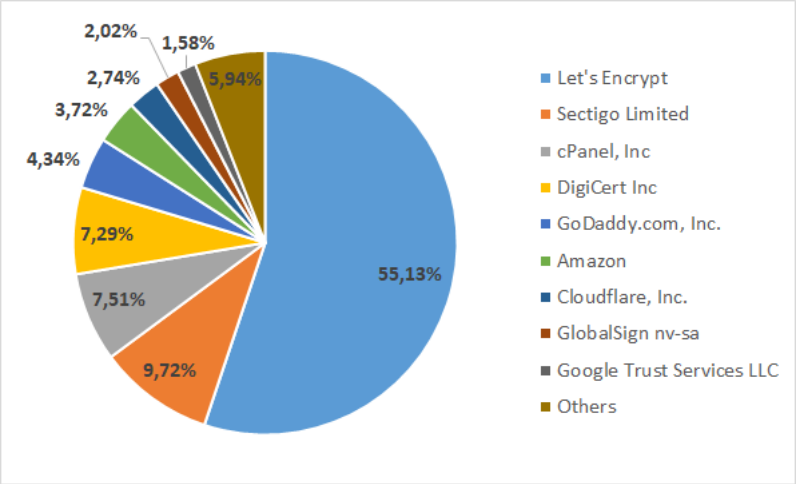


Figure 4.3: Percentage of issued certificates per CA

Authority Key Identifier Extension In order to determinate how many certificates support the Authority Key Identifier extension, I wrote a python script which try to read extension of each certificate separately for the leaf and CA set. The results show that 99,77% of certificates (440.172) contains the Authority Key Identifier extension and only 0,23% (1031) do not contain the extension. In the CA set, instead the 87,13% of certificates contains the AKI extension and the 12,87% does not support it. All certificates, in both set, have marked the extension as non critical and only one certificate (in the leaf set) raised a value error exception.

Subject Key Identifier Extension Subject Key Identifier extension is present in the 99,74% of leaf certificates and in the 95,21% of CA certificates. In both cases, in all certificates, the extension is marked as non critical and only one certificate in the leaf set has raised a value error exception.

Key Usage Extension Key Usage extension is present in the 99,19% of leaf certificates and in the 52,88% of CA certificates. In the 99,90% of leaf certificates the extension is marked as critical while 8,56% of CA certificates contains the extension marked as non critical.

Extended Key Usage Extension Extended Key Usage extension is present in the 99,43% of leaf certificates and only in the 24,49% of CA certificates. In all leaf CA certificates the extension is marked as non critical, while in the leaf set only in 112 certificates is marked as critical.

Certificate Policies Extension Certificate policies extension is present in the 99,08% of leaf certificates and in 38,52% of CA certificates. In all certificates (leaf and CA set) the extension is marked as non critical.

Subject Alternative Name Extension The 99,79% of certificates belonging to the leaf set specify Subject Alternative Name extension, while the 87,00% for the CA set. In all leaf and CA certificates the extension is marked as non critical, while only one certificate belonging to the leaf set has the extension marked as critical.

Expiration date I have also checked the certificates expiration date for both sets against the 26th of August (date in which I started downloading certificates). Results in figure 4.4 show that only the 1,44% of collected certificates belonging to the leaf are expired; on the contrary, surprisingly, the 20,67% of collected CA certificates are expired.

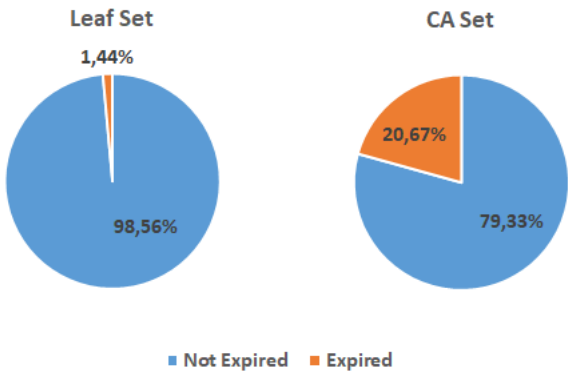


Figure 4.4: Percentage of expired/non expired certificates for leaf and CA set

SCT extension SCT extension contains the proof, generated by public logs, that the certificate where it appears will be logged into a public log as soon as possible. The 98,93% (436.496) of certificates belonging to the leaf set contains SCT extensions and among these the 80% list 2 SCTs, the 19,06% list 3 SCTs and the 0,84% more than 3 SCTs. Certificates belonging to CA set do not contain this extension.

4.1.1 Extensions for checking revocation status

CRL Distribution Points Extension This extension has a crucial role since it is one way for checking revocation status of the certificate where appears. CRL distribution point is present only in the 40,37% of CA certificates and only in the 32,54% of leaf certificates. For the leaf set, the preferred method for checking revocation status is the OCSP since OCSP uri appears in 99,08% of certificates (inside AIA extension).

In order to establish how many certificates of the leaf set that support CRL have been revoked, I wrote a python script that download the CRL and check the certificate status (for the ones that contain CRL distribution point extension). Results shows that 278 out of 143.550 certificates, that list CRL distribution point extension, have been revoked while 142.335 are in good state; 937 certificates failed in downloading the CRL.

Authority Information Access Extension Authority Information Access extension appears in the 99,08% of leaf set certificates, while it is less popular in the CA certificate appearing only in the 38,42%. In both case, in all certificates, the extension is marked as non critical and only one certificate in the leaf set has raised a value error exception.

As for CRL, in order to establish how many certificates of the leaf set, that lists a potential OCSP responder, have been revoked I wrote a bash script that makes a request to OCSP responder reachable through the link present in AIA extension for each certificates that present AIA

extension. 600 certificates belonging to the leaf set have been revoked, 409.435 are in good state and 11 have an unknown state. OCSP responders of the remain part of certificates, cannot be contacted due to a connection error.

As showed in figure 4.5 OCSP is the preferred method for checking revocation status in the leaf set since a potential OCSP responder appears in the 99,08% of certificates (437.143). Only 6 certificates without the AIA extension list a potential CRL distribution point while the 0,92% (4054 certificates) does not provide a way for checking revocation status (neither a CRL distribution point neither a potential OCSP responder). The 32,54% of certificates (143.550) belonging to the leaf set list a potentially CRL distribution point and it is almost the same number of certificates (143.544) that support both methods (this means certificates that choose CRL as method for revocation checking contain also a potential reachable OCSP responder); while the 66,54% of certificates (293.599) list a potential OCSP responder but does not provide a CRL distribution point.

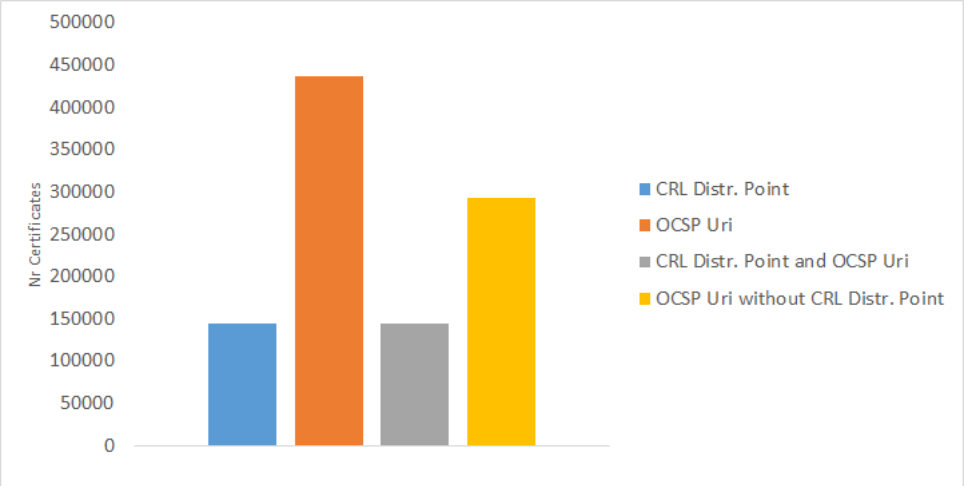


Figure 4.5: Revocation status for leaf certificates

As depicted in figure 4.6, OCSP has been chosen only by the 38,42% of CA certificates (433). Only 1 certificate is embedded with a potential OCSP responder but does not list a CRL distribution point, while the 59,54% (671 certificates) does not provide a way for checking revocation status (neither a CRL distribution point neither a potential OCSP responder). The 40,37% of CA certificates list a potential CRL distribution point and the 38,3% contains also a potential reachable OCSP responder; while only the 2,04% (23 certificates) provide a CRL distribution point without a potential OCSP responder.

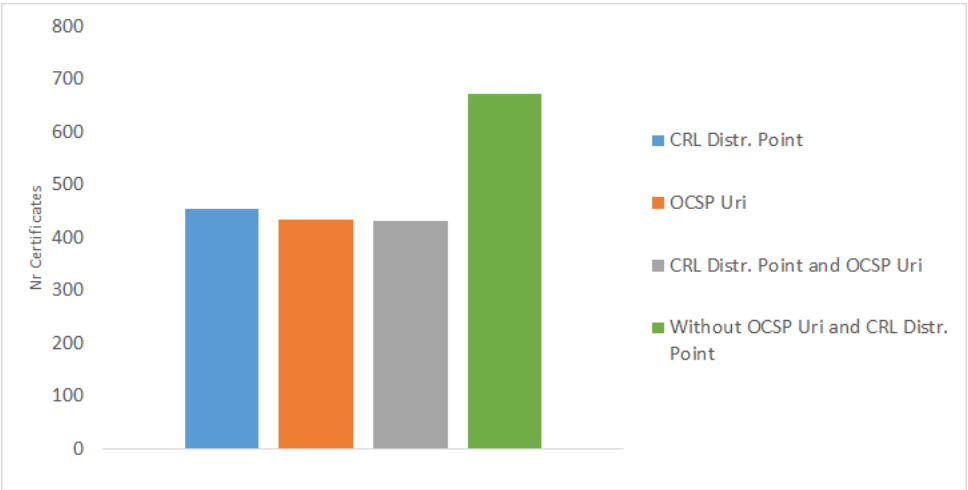


Figure 4.6: Revocation status for CA certificates

4.2 Certificates status check

4.2.1 Checking certificate status against OCSP

I have downloaded certificate advertised by `www.polito.it` and I have inspected its content with openssl x509 command 4.7

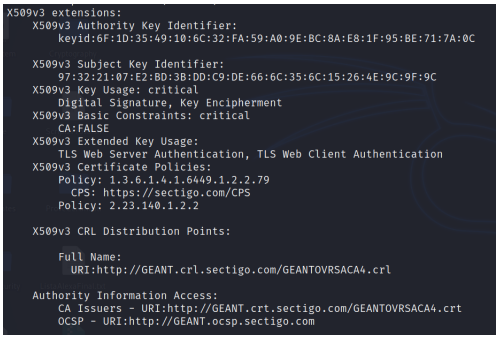


Figure 4.7: Certificate of `www.polito.it` web server inspected with openssl

Then I have downloaded the certificate of its issuer and converted it into PEM format 4.8.

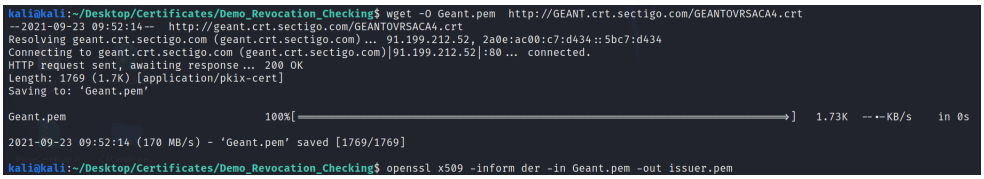


Figure 4.8: Download of `www.polito.it` issuer certificate and conversion into PEM format

At this point I have checked the status of polito certificate against OCSP through the URL present in the AIA extension visible in figure 4.7. For this purpose I have used openssl ocsp command 4.9. The certificate is in good state and it has not been revoked.

In order to analyse OCSP response in case of a revoked certificate, I downloaded also a revoked certificate from the website `https://revoked-rsa-dv.ssl.com/` and after having inspected its content with openssl, I have downloaded the certificate of its issuer and took note of the OCSP URI present in AIA extension. At this point I had all stuffs needed for checking the certificate status against OCSP.

The OCSP response in picture 4.10 suggests that the certificate has been revoked on 17/06/2021 and that on 30/09/21 there will be newer information available about the certificate status. The response contains also other information such as: the version of OCSP protocol (version 1 value 0X0), OCSP responder ID (7D4FE8D455E2870BD0E4A1B4AAA55693B5A6D6BF), date and time of computed response (23/09/2021 14:20) and other fields for identifying the certificates (Issuer Name Hash, Issuer Key Hash and Serial Number).

4.2.2 Checking certificate status against CRL

I have also checked the status of polito certificate against CRL by using openssl crl command. I downloaded the CRL from the URL indicated in CRL distribution point in figure 4.7 and then I have converted it into PEM format 4.11

In order to check validity of CRL, I run openssl command for verifying CA signature over it and with the second command in figure 4.12 I verified the certificate including revocation checking against CRL. Verification ends without any problem.

As depicted in picture 4.13, I have repeated same actions for the revoked certificate of the web site `https://revoked-rsa-dv.ssl.com/`:

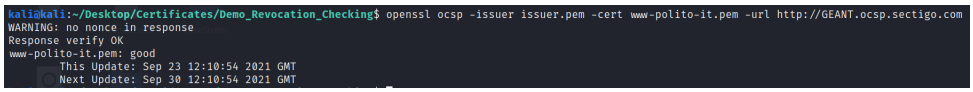


Figure 4.9: Check status of `www.polito.it` certificate against OCSP

Figure 4.10: Check status of a revoked certificate against OCSP

- Download the CRL from the URI indicated in CRL Distribution Point extension;
- Convert CRL from DER to PEM format;
- Verify signature over CRL;
- Verify validity of certificate, including revocation status. As figure 4.13 suggests, the certificate has been revoked and verification failed.

4.3 Inspection of some revoked certificates

In figure 4.14 I have inspected the content of a revoked certificate with openssl command: the website advertises this certificate is `laga.se`.

The certificate analysed in figure 4.14 with openssl has a different serial number from the one advertised by visiting `laga.se` with Google Chrome on the 29 of September.

In order to check the validity of newly certificate advertised by `laga.se` and showed in 4.15, I downloaded it together of its issuer certificate and the CRL. As figure 4.16 suggests, the certificate is valid.

An example of revoked certificate, found querying an OCSP responder, is one advertised by `ideam.gov.co`. As depicted in figure 4.17 with openssl s_client -connect command verification goes well because it does not check revocation status of certificates in the chain.

GNU TLS library instead give you possibility for checking also revocation status with ocsp. By adding -ocsp flag, figure 4.18 depicts how GnuTLS verification fail due to revocation of certificate.

4.4 OCSP Stapling checking

In order to check whether `www.amazon.com` and `www.youtube.com` servers support OCSP stapling, I used Openssl s_client command for establishing a TLS connection with both web site. As showed in figure 4.19 amazon web server support OCSP stapling since the OCSP response is populated with revocation information, while YouTube one does not support OCSP stapling as OCSP response suggests in figure 4.20.

I replicated this check for all sites listed in AlexaTop1M file, in order to know how many web servers today support OCSP stapling. Surprisingly the percentage of web server that support OCSP stapling has grown in the last years: 45,39% of servers present in the list support OCSP stapling against the 2,60% of 6 years ago [3].

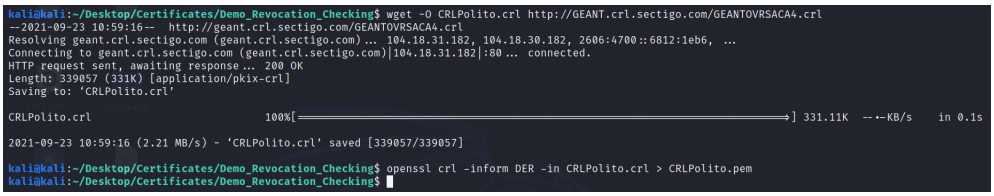


Figure 4.11: Command run for downloading a CRL in DER format and converting it in PEM

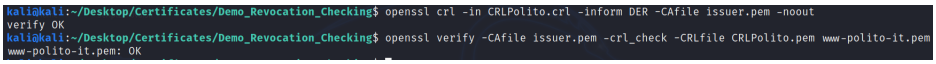


Figure 4.12: Command run for checking signature over CRL and for verifying www.polito.it certificate including revocation checking

Figure 4.13: Command run for downloading a CRL; for converting it into PEM format; for verifying signature over it; for validating certificate including revocation checking

Figure 4.14: Example of revoked certificate inspected with openssl

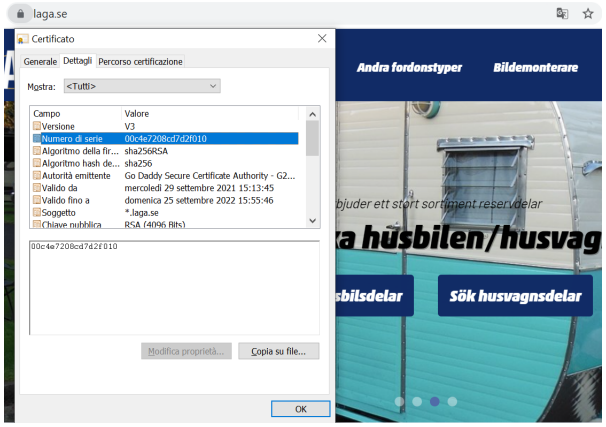


Figure 4.15: Certificate advertised by laga.se

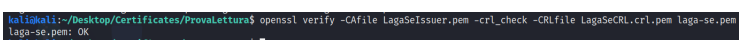


Figure 4.16: Openssl command for checking the status of certificates advertised by laga.se on 29 of September

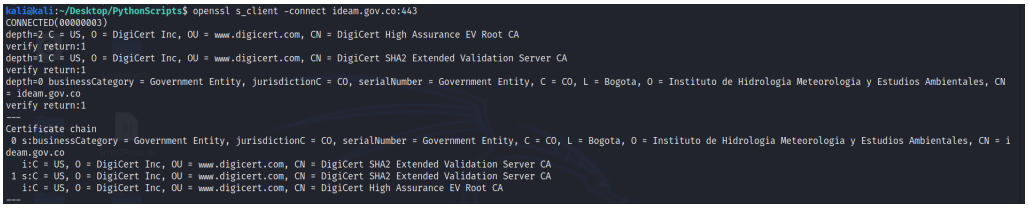


Figure 4.17: Openssl command for establishing a TLS connection with ideam.gov.co

Figure 4.18: GnuTLS command for establishing a TLS connection with ideam.gov.co

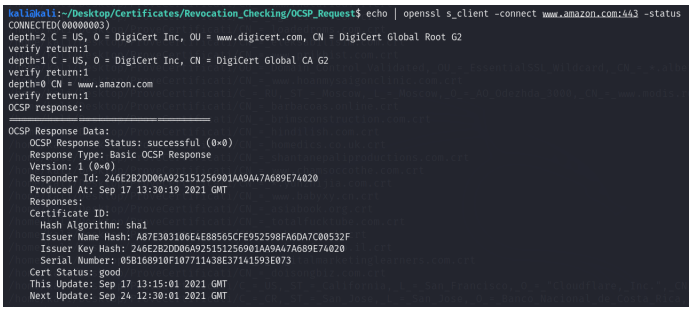


Figure 4.19: Command run for checking whether amazon web server support OCSP stapling: it supports the functionality since the response contains revocation information

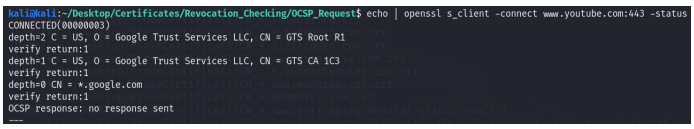


Figure 4.20: Command run for checking whether YouTube web server support OCSP stapling: it does not suppor the functionality since the response contains "no response set"

Chapter 5

Web browsers behaviour on handling revocation information

Main goal of this study is to check whether web browsers handle correctly certificates revocation information during validation process. In this section I describe the approach followed and the implemented setup for testing browsers behaviour in the validation of certificates. As first things I list the browsers chosen for the tests specifying the version and the OS under which they are executed. Secondly, I describe the kinds of invalid certificates I will use to test browsers behaviour together with their generation process. Then I describe how I have set up OCSP responder and servers for downloading CRL information. Finally, I perform some tests, under different conditions, to give a view of how web browser validate certificate chains and how they manage certificate revocation information.

Since there are not specific guidelines on how browsers should perform certificate validation process, browsers developers are responsible of implementing the process from sketch and they should not take care about some fundamental aspects in the validation process as checking revocation information. The goal of this study is to understand how web browsers perform certificate validation process before establishing a secure connection paying attention on how they handle revocation information. In order to study the browsers behaviour in validating certificates, there are some preliminary steps to perform:

- List the browsers will be used in the study: nowadays there are a large number of browsers for each platform. It is necessary to choose the most used ones together with the platform under which they are executed. I treat it in 5.0.1;

- Identify set of test cases that cover typical situation in which browsers should find into and study their behaviours. Test cases include chains revoked certificates at each level of chain and some network related problem (e.g OCSP responder or CRL server not available). I choose to use chains of three certificates because they are enough to test main classes of certificates (root CA, intermediate CA and leaf). Tests include standard DV certificates and also EV certificates. They are summarized in table 5.2;

- In order to implement test cases in table 5.2, I need to generate some certificates and CRLs, but also I need to implement an OCSP responder that manages OCSP queries. I describe the process of certificate and CRL generation in section 5.0.2;

- Test cases require to install certificate chains on a server with which browsers will try to connect to. For each test I install the target chain on the server and then I try to establish a TLS connection with chosen browsers. I describe the process of server configuration in 5.0.4.

5.0.1 Target browsers and platform

The number of browsers avaiable for surfing on internet grows exponentially in the lasts years and nowadays users can choose from a large variety of browsers for accessing on Internet. Their behaviour change depending on the OS where they run, and for this reason I choose to replicate the tests for each browsers under different OS (when they have a compatible version). The process of selection of web browsers is based on real data, coming from Netmarket-share [19] and update at 22 October 2021. Data show that Google Chrome is the most used browsers among users with a percentage of 69,28%, followed by Edge (7,75%), Firefox (7,48%) and Internet Explorer (5,21%). For my tests I decide to use browsers leaders in the browser space: Chrome, Mozilla, Edge, Internet Explorer, Safari and also Opera which is a browsers particularly used from gamers. Table 5.1 summarizes the browsers chosen for conducting tests together with the OS under which they will be run.

Browser	Version	Operating System (Platform)
Google Chrome	95.0.4638.54	W/OSX/L
Mozilla Firefox	93.0	W/OSX/L
Opera	80.0.4170.63	W/OSX/L
Internet Explorer	20 H2	W
Microsoft Edge	95.0.1020.30	W
Safari	13.1.2	OSX

Table 5.1: List of browsers to use for conducting tests together with platform under which they will be run

5.0.2 Certificates, CRLs and OCSP process generation

For executing tests listed in table 5.2, I have to generate some certificates and chains. Each chain is composed of 3 certificates: a self-signed root CA certificate, an intermediate CA certificate issued by the root CA and a leaf certificate issued by the intermediate CA. In this way I impersonate the role of a root CA, an intermediate CA and a normal user which are critical for studying validation process. Additionally I created CRLs related to CA and leaf certificates; I also implemented an OCSP responder. In order to generate and manage certificates I used OpenSSL 1.1.1l stable version, that is the most common library for generating, signing and validating certificates. I use 2020.4 version of Kali running on a Virtual Machine hosted by VM Aware tool.

OpenSSL configuration OpenSSL uses a configuration file `openssl.cnf` to handle the contents of X509 certificates. In order to satisfy requirements of this work, I created four different copies of `openssl.cnf` file for EV and non EV root and intermediate certificates. I created two directories `/rootCA/` and `/intermediateCA/` respectively for root CA and intermediate CA: each one contains its configuration file. OpenSSL configuration file contains a sort of "profiles" useful for managing the content of leaf and CA certificates will be issued using that configuration file: usr_cert, v3_ca and v3_OCSP. By default usr_cert profile defines the content of leaf certificates (e.g. basicConstraint extension set to false); on the contrary v3_ca profile defines the content of CA certificates (e.g. basicConstraint extension set to true); instead I define v3_OCSP profile to define the content of OCSP responder certificates. This allow me to generate certificates under different conditions only by changing and adding some fields in the configuration file (e.g. adding crlDistributionPoints and authorityInfoAccess extensions). These are lines of `openssl.cnf` file containing definition of usr_cert, v3_ca profiles an v3_OCSP:

Listing 5.1: OpenSSL configuration file used for generating certificates

```
...
[ usr_cert ]
basicConstraints=CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

```
nsComment   = "OpenSSL Generated Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
authorityInfoAccess = OCSP;URI:http://192.168.0.109:49600
crlDistributionPoint = URI:http://192.168.0.109:49601
subjectAltName=@alt_names

[ alt_names ]
IP = 192.168.0.100:49500
DNS = certificate-check.it

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = critical,CA:true
keyUsage = cRLSign, keyCertSign
authorityInfoAccess = OCSP;URI:http://192.168.0.109:49600
crlDistributionPoint = URI:http://192.168.0.109:49601

[ v3_OCSP ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = OCSPSigning
```

Generation of root CA key and certificate Configuration file 5.1 is used to generate an asymmetric key pair and associated X509 certificate for the root CA. To accomplish this task I use these commands:

```
$ openssl genrsa -out rootCA/rootCA.key 2048
$ openssl req -new -x509 -days 365 -key rootCA/rootCA.key -outrootCA/
    rootCA.crt -config rootCA/validation.cnf
```

With the above commands I create a self-signed root CA with configuration specified in the v3_ca section of `validation.cnf`. The created certificates expires after 365 days and the command generates `index.txt` file containing the database of certificate directly signed by root CA and a `serial` file. Every time the root CA signs or revokes a certificate (CA or leaf), an entry is added into the `index.txt` file and it contains: status (R-revoked or V-valid), expiration date, revoked date, serial number, certificate file name and subject name. The `serial` file is used by the root CA for storing the serial number of the next certificate to be signed and its content change every time a new certificate is issued (increments by one).

Generation of intermediate CA key and certificate Since intermediate CA certificate must be signed by root CA, commands for its generation are different from ones used for generating root certificate. For this reason I use opens SSL for generating an asymmetric key pair and a Certificate Signing Request (CSR) by means of the following command:

```
$ openssl req -new -newkey rsa:2048 -nodes -keyout
    intermediateCA/intermediateCA.key -out
    intermediateCA/intermediateCA.csr
```

CSR is now sent to the root CA which will sign it and return it back to the intermediate CA. I use the following command for signing the CSR:

```
$ openssl ca -config validation.cnf -keyfile rootCA.key -cert rootCA.crt
    -policy policy_anything -extensions v3_ca -notext -in
    ../intermediateCA/intermediateCA.csr -out
    ../intermediateCA/intermediateCA.crt
```

The `index.txt` file of root CA now contains one entry which refers to the already signed certificate.

```
V 221015152313Z  0A unknown
    /C=IT/ST=Caltanissetta/L=Caltanissetta/OU=Intermediate
    CA/CN=IntermediateCA/emailAddress=intermediateca@gmail.com
```

Since the created certificate is CA certificates, a `index.txt` and `serial` files are created in the intermediateCA/demoCA stores for managing the leaf certificates it will issue. In order to verify whether intermediate CA certificate has been correctly generated I use the following command:

```
$ openssl verify -CAfile rootCA/rootCA.crt intermediateCA/intermediateCA.crt
```

Generation of leaf key and certificate Steps needed for generating leaf certificates are similar to the ones used for generating intermediate CA certificates. As first thing, I generate an asymmetric keypair and a Certificate Signing Request (CSR) for the leaf certificate as follows:

```
$ openssl req -new -newkey rsa:2048 -nodes -keyout Leaf1/leaf1.key -out
    Leaf1/leaf1.csr
```

The generated CSR is now sent to the intermediate CA which will sign it and return it back to the requestor. I use the following command for signing the CSR:

```
$ openssl ca -config validation.cnf -keyfile intermediateCA.key -cert
    intermediateCA.crt -policy policy_anything -extensions usr_cert
    -notext -in ../Leaf1/leaf1.csr -out ../Leaf1/leaf1.crt
```

5.0.3 Leaf, intermediate CA and root CA server configurations

The intermediate CA signs the leaf CSR with `usr_cert` profile because it is a non CA certificate and to verify the correctness of above process I run the command:

```
$ openssl verify -CAfile cat <(intermediateCA/intermediateCA.crt
    rootCA/rootCA.crt) Leaf1/leaf1.crt
```

	CRL	OCSP
Root CA	CRL Not Available	OCSP Not Available
	Certificate Revoked	Certificate Revoked
Intermediate CA	CRL Not Available	OCSP Not Available
	Certificate Revoked	Certificate Revoked
Leaf Certificate	CRL Not Available	OCSP Not Available
	Certificate Revoked	Certificate Revoked

Reject unknown status
Turn to CRL when OCSP fails

Table 5.2: List of test cases to perform under different platform, using different web browsers

Certificate revocation: CRLs generation and OCSP responder implementation In order to execute some test cases listed in table 5.2 I need to revoke certificates previously issued. In order to revoked the certificate, I use the following command:

```
$ openssl ca -keyfile intermediateCA.key -cert intermediateCA.crt -config
    validation.cnf -revoke ../Leaf1/leaf1.crt
```

After the execution of this command the entry of `index.txt` corresponding to the target certificate changes its value: V is replaced by R. In order to publish revocation information, I can create a CRL and host it at the URL specified in `validation.cnf` file. I use openSSL command for generating CRL:

```
$ openssl ca -keyfile intermediateCA.key -cert intermediateCA.crt -config
    validation.cnf -gencrl -out CRL/crl.crl
```

Since the generated CRL is in PEM format, I run the following command for converting it into DER format before hosting it at the URL:

```
$ openssl crl -inform PEM -outform DER -in CRL/crl.crl -out CRL/crl.der
```

In order to set up a OCSP responder at the url specified in the `validation.cnf` I need to generate an asymmetric key pair for the OCSP server, a CSR and then send the CSR to the root CA and signs it with the key of root CA. I generates the OCSP responder certificate with `v3_ocsp` profile that contains basic constraint extension set to false, and extended key usage contains OCSP signing value.

```
$ openssl req -new -newkey rsa:2048 -nodes -keyout
    intermediateCA/OCSP/ocspSigning.key -out
    intermediateCA/OCSP/ocspSigning.csr
$ openssl ca -config validation.cnf -keyfile intermediateCA.key -cert
    intermediateCA.crt -policy policy_anything -extensions v3_OCSP
    -notext -in OCSP/ocspSigning.csr -out OCSP/ocspSigning.crt
```

EV certificate generation In order study the behaviour of browsers when manage EV certificates, `validation.cnf` file may be changed. The CA that will issue the EV certificates must be able to sign this kind of certificates, and it is possible by adding a special OID value into the certificate. Most popular CAs disseminate OID values, so that they can be used by web browsers developer for identifying EV certificates. In order to emulate the behaviour of a CA authorized in signing EV certificates, I append DigiCert Oid value [20] in the new configuration file named `validationEV.cnf` instead of creating a new random OID and add it into browser source code.

The new configuration file, `validationEV.cnf`, contains some additional lines such as the Oid presented above:

```
[ new_oids ]
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
...
```

Finally EV certificates can be generated using the new configuration file, following the commands presented in the paragraph above. Since *policy_match* policy obliges certificates to have values equal to the ones of its issuer certificate, I use *policy_anything* policy which allows certificates to accept any values in the subject field without any kind of restrictions.

5.0.4 Experimental setup

Before running tests, it is important to set up the experimental testbed and validate it. I choose to use a windows PC for running the leaf server on the port 49152 (supporting https), where I will install a variety of chains for performing tests. I also installed Wireshark on the Windows PC to record network traffic from the browsers: this is useful to understand whether browsers perform a request or not (e.g. it helps to understand whether a browsers perform a OCSP request or http get for downloading a CRL). In order to perform tests from Windows I use the same machine where the server run, by using Mozilla, Chrome, Opera, Edge and IE. While for performing test from Unix OS, I use Kali 2020.4 running on a virtual machine where I installed Mozilla, Chrome and Opera. Finally for launching test from Mac OS, I use another computer always connected in the same LAN where I installed Mozilla, Chrome, Opera and Safari. In each desktop platform I configured DNS local file for resolving the `certificate-check.it` with the local ip (e.g for windows platform it is 192.168.0.100).

Leaf server configuration I configured a leaf server to use for testing browsers behaviour with Internet Information Service of windows. For this purpose, on windows machine I configured a https server at the url `certificate-check.it:49152` where, once per time, I will install different chains depending on the test I would like to conduct. Picture 5.1 shows Internet Information Service configuration page that lists services running on the machine: leaf server is running on `certificate-check.it:49152`. Certificates generated with openSSL are by default in PEM format, while Internet Information Service require certificates and private keys in PKCS#12 standard. For this reason, I convert them in PKCS#12 format with the following openSSL command:

```
$ openssl pkcs12 -export -out leaf1.pfx -in leaf1.crt -inkey leaf1.key
```

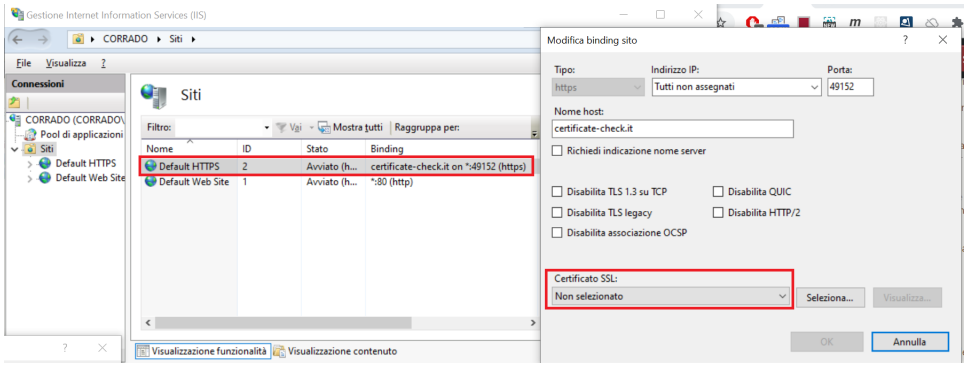


Figure 5.1: Internet Information Service configuration for leaf server at the url certificate-check.it:49152

As showed in picture 5.1, it is possibile to choose the SSL certificate to install on the server in an easy way.

Root and intermediate certificates installation Root CA certificate and intermediate CA certificate need to be installed on the browsers trusted root and intermediate stores so that browsers can trust that CAs. In order to install them on Chrome and Firefox under Windows, I open the browser, select settings, security, handle certificates and import the certificates with the apposite command.

CRL service and OCSP responder CRLs and OCSP are the common ways with which browsers retrieve revocation information about the certificates presented by the web server they are connecting to. For CRL checking, browsers read the url present in CRL distribution point extension and perform a http get for downloading the CRL. On the other side, CA must publish CRLs in public and available repositories: for this reason I configure a service which allows to download updated version of intermediateCA CRLs with the following command:

```
$ python3 -m http.server 49501
```

I run this command inside the folder that contain CRLs, so that they can be easily downloaded from hosts inside the same LAN.

For OCSP checking, instead, I configured an OCSP responder at the url specified in AIA extension of leaf certificate by means of the following command:

```
$ openssl ocsp -index demoCA/index.txt -port 49500 -rsigner
    OCSP/ocspSigning.crt -rkey OCSP/ocspSigning.key -CA
    intermediateCA.crt -text -ignore_err
```

Now the OCSP responder is ready for handle OCSP queries coming from network: when receives an OCSP request, it reads the target certificate and looks for corresponding entry in the `index.txt` file. It elaborates the response depending on the value of the corresponding entry in the `index.txt` file (R or V). In case there is not an entry in the file corresponding to the target certificate, the response status will be unknown.

5.0.5 Testbed validation

Finally I have to validate the implemented testbed to be sure that everything works as expected. Validation of our implemented infrastructure consists on:

1. Checking whether browsers built and validate correctly certificate chain;

2. Checking whether CRLs and OCSP server are really available and allow browsers to download list of revoked certificates or sends OCSP requests.

This is a critical step in my study because if something fails, assumptions made until now are wrong and it is not possible to study browsers behaviour on handling certificates. In order to validate the built infrastructure, I generated a valid (not expired, not revoked and signed by a trusted CA) certificate following the commands explained in section 5.0.2 and I installed it in a service hosted by the port 49512 as described in 5.0.4. For testbed validation I use Internet Explorer which always tries to download CRL from url indicated in CRL distribution point of leaf certificate and Mozilla Firefox that perform an OCSP request to the url specified in AIA extension as first action for retrieving revocation information. In this way, I can test the effectiveness of my testbed and in particular check whether the OCSP responder and service to disseminate CRLs are available and works as supposed. I start services that host CRL and OCSP responder as described in 5.0.4, I active service where I installed the certificate chain and before connecting I cleared windows cache with the following command:

```
$ certutil -urlcache * delete
```

In order to check whether browser is able to download CRL and make a OCSP request to the OCSP responder I am capturing network traffic with Wireshark.

CRLs availability As first thing I try to establish connection with `certificate-check.it` when OCSP responder and CRL are both available and I have recorded network traffic with Wireshark. For this purpose I use Internet Explorer that will perform an OCSP query for each certificate in the chain as first option and turn to a CRL check in case of an error related to the OCSP responder. Since the OCSP request performed by Internet Explorer fails due to it was not well formatted for the implemented OCSP responder and since OCSP responder has be run without -ignore_err extension, it turns to CRL check by downloading CRL from url present in certificate CRL Distribution Points extension.

No.	Time	Source	Destination	Protocol	Length	Info
781	8.335667	192.168.0.100	104.18.24.243	HTTP	300	GET /MFQwUjBQME4wTDAJBgUrDgMCGgUABBSjA8CoiHvUecQnjrXX…
2509	25.484052	192.168.0.100	192.168.0.109	HTTP	274	GET /MEIwQDA%2BMDwwOjAJBgUrDgMCGgUABBSzQlY4dnUm1Y2axC…
2565	27.518109	192.168.0.100	192.168.0.109	HTTP	181	GET /crl.crl HTTP/1.1
2642	28.082769	192.168.0.100	93.184.220.29	HTTP	294	GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBQ50otx%2Fh0Ztl%2Bz…
3130	51.091924	192.168.0.100	92.122.95.146	HTTP	334	GET /msdownload/update/v3/static/trustedr/en/authroot…

Figure 5.2: Network traffic analysed with Wireshark while connecting to `certificate-check.it:49152` with Internet Explorer

As shown in figure 5.2, Internet Explorer perform a http get for the file crl.crl and then, as depicted in figure 5.3, loads correctly the page without any warning.

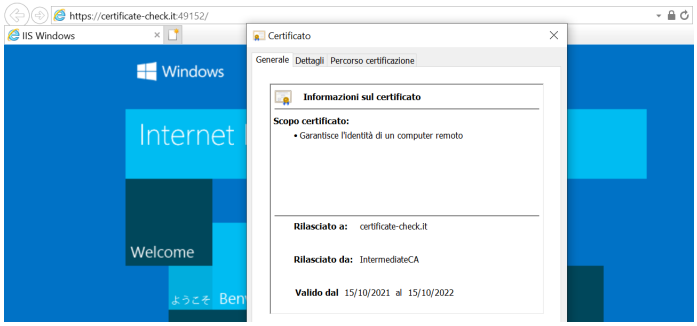


Figure 5.3: Page of the `certificate-check.it:49152` correctly loaded from Windows Explorer

This test proves that CRL has been correctly downloaded from url indicated in CRL distribution point extension and the configuration works correctly. Browsers correctly reads url present in CRL extension and perform an http get to the server hosting CRLs. As figure 5.4 shows, server hosting CRL receives a get request from the ip address 192.168.0.100 and correctly allows CRL downloading.

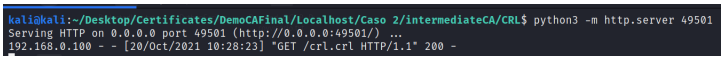


Figure 5.4: Http get request received from the server hosting CRL file

OCSP responder availability Secondly I try to establish a connection with `certificate-check.it` when OCSP responder and CRL are both available, for checking OCSP responder availability. I have always recorded network traffic with Wireshark and this time I used Mozilla Firefox which performs OCSP request to the url specified in leaf certificate AIA extension, as default way for retrieving revocation information, in a correct format for the implemented OCSP responder.

No.	Time	Source	Destination	Protocol	Length	Info
2201	36.204389	192.168.0.100	192.168.0.109	OCSP	486	Request
2207	36.207190	192.168.0.109	192.168.0.100	OCSP	297	Response

Figure 5.5: Network traffic analysed with Wireshark while connecting to `certificate-check.it:49152` with Mozilla Firefox

Figures 5.5 shows that Mozilla Firefox sends an OCSP request to the OCSP responder, and OCSP server answers with an OCSP response. Browser loads correctly the web page in the same way as Internet Explorer loads it in figure 5.3. This test proves that OCSP responder is available at the url specified in the leaf certificate AIA extension and that the configuration works correctly.

These tests validate the testbed configuration: servers which host CRLs and OCSP responders are both avaiable and work as expected. They can be contacted by any host connected in the LAN for retrieving revocation information about each certificates appearing in the installed chain.

5.1 Results

This study aims to analyse behaviour of most recent web browsers in validating certificate chains, focusing on how browsers handle certificate revocation checking under different conditions: how browsers behave when OCSP server is not available or when server that host CRL is not ready available? Does browsers refuse connection in case they are not able to retrieve revocation information about each certificate in the chain? In order to study browsers behaviour I perform tests listed in table 5.2 using Chrome, Mozilla and Opera under OSX, Windows and Linux operating system while I use Internet Explorer and Microsoft Edge under Windows OS and Safari under OSX.

Google Chrome

Chrome exploits NSS library [21] for establishing TLS connection and platform-dependent library for certificate validate as reported in the documentation of the open source Chromium project [22] where Chrome came from. Chrome behaves differently under Windows, OSX and Kali Linux in validating EV and non EV certificates.

On Windows, for non-EV certificates, it does not retrieve any revocation information: OCSP queries and requests for downloading CRLs are not performed by Chrome. Tests show that browser accepts the connection with experimental server when a non-EV revoked certificate is present in the chain (at any level). For EV certificates, instead, Chrome performs OCSP requests for checking revocation status of each certificate in the chain. It first sends OCSP requests by means of HTTP Get which fails since they are not supported by the implemented OCSP responder and then re try with HTTP Post which is format expected by the OCSP responder. Chrome request CRL in presence of *unknown* response from the OCSP responder, for each certificate in the chain; while refuse connection when at least one revoked certificate is present in the chain. If OCSP responder is not avaiable, Chrome performs a request for downloading CRL.

On OSX for non-EV certificates, as for Windows, it does not retrieve any revocation information by means of OCSP queries or requests for CRL download. This lead on accepting connection

with set up server when a non-EV revoked certificates is present in the chain (at any level). For EV certificates, instead, Chrome performs OCSP requests for checking revocation status of leaf and intermediate CA certificates, but since they are encapsulated inside an HTTP GET, OCSP responder answer with a malformed status response. Differently from Chrome in Windows, on OSX it does not try with HTTP posts and does not fall to CRLs. This lead on accepting connection in presence of a revoked certificate in chain (at any level) installed on the server Chrome tries to connect to. Since OCSP requests fail because they are performed through HTTP Gets, it is not possible testing Chrome behaviour when receives *unknown* status from OCSP responder.

On Kali Linux for non-EV certificates and EV-certificates, Chrome does not request revocation information neither through CRL and neither through OCSP. Tests show that it accepts the connection when a chain with a revoked certificate (at any level) is installed in the experimental server.

In case both OCSP responder and CRL are not available, Chrome silently accepts the connection for EV and non-EV certificates under each platform.

Mozilla Firefox

Firefox, as Google Chrome, exploits NSS library [21] and its behaviour is consistent in each OS: it behaves in the same way in Kali Linux, Windows and OSX for EV and no-EV certificates. It queries only OCSP responder for retrieving revocation information of leaf certificate, and it does not take care of the other certificates in the chain (rootCA and intermediateCA certificates). This lead on refusing connection when a revoked leaf certificate is present in the chain installed on the server; while on accepting connection whether the intermediate or root CA certificate is revoked. In case the OCSP responder is not avaiable, firefox does not fall to CRL and silently accepts the connection; while in presence of an *unknown* response from the OCSP responder, it correctly shows a security warning.

Opera

Opera born from Chromium project [22] and behaves differently in each OS system for non-EV and EV certificates. On Windows, for non-EV certificates, it does not fetch revocation information neither from OCSP and neither from CRL. For this reason, tests show that Opera accepts the connection with the experimental server in presence of a revoked certificate in the chain (at any level). For EV certificates, instead, Opera performs OCSP requests for retrieving revocation information about rootCA, intermediateCA and leaf certificates. Since these requests are performed through HTTP Gets, OCSP responders answer with malformed status request. From these responses, Opera automatically will retry to get revocation information through HTTP posts. Opera correctly refuses connection with demo server in presence of a revoked certificate in the chain; it falls back to CRL in case OCSP responders are not available and requests CRL whether receives an *unknown* response status from an OCSP responder.

On OSX, for non-EV certificates, it does not check revocation status of certificates installed in the server want to connect to. This leads on accepting connection with experimental server when at least one certificate (at any level) in the installed chain has been revoked. Also for OSX, Opera treats EV certificates differently: it performs OCSP requests for retrieving revocation information related to leaf and intermediate CA by means of HTTP Gets, which are not acceptable by the implemented OCSP responder. From this, it does not requests OCSP through HTTP posts as it makes on Windows and it does not requests CRLs. For this reason it accepts connection without displaying any security warning with the experimental server when a revoked certificate is present in the installed chain (at any level). Since OCSP requests fail because they are performed through HTTP Gets, it is not possible testing Opera behaviour when receives *unknown* status from OCSP responder.

On Kali Linux for non-EV certificates and EV-certificates, Opera does not request revocation information neither through CRL and neither through OCSP. Tests show that it accepts the connection when a chain with a revoked certificate (at any level) is installed in the experimental server.

Internet Explorer

Internet Explorer treats non-EV certificates in the same way of EV certificates and before establishing a connection fetches revocation information of intermediateCA and leaf certificates. Tests shows that Internet Explorer first requests revocation information regarding leaf and intermediateCA certificates to OCSP responders with HTTP Gets, that fail for the malformed format of the request. From these, it automatically performs HTTP Posts which are correctly served by the implemented OCSP responders. Internet Explorer detects when on experimental server is installed a certificate chain that contains at least one revoked certificate (at any level) and it correctly refuses connection displaying a security warning. Internet Explorer fails on fetching revocation information related to root certificate: it silently accepts connection with the experimental server when the root certificate of the installed chain has been revoked. This happens because Internet Explorer limits on retrieving revocation information only related to leaf and intermediates CA certificates. In case OCSP responders are not available, it fall back to CRLs; while it requests CRL when receives *unknown* status from OCSP responder. In case both OCSP responder and CRL are not available, Internet Explorer silently accepts the connection for EV and non-EV certificates.

Microsoft Edge

Microsoft Edge, as Google Chrome and Opera for Windows, treats non-EV certificates differently from EV certificates. For non-EV certificates, Microsoft Edge does not perform OCSP requests and does not download CRLs for retrieving revocation information of certificates belonging to the chain installed on the experimental server it wants to connect to. It fails on detecting web sites served by revoked certificates: tests show that it silently establish connection with the experimental server when at least one revoked certificate is present in the installed in the chain (at any level). For EV-certificates, instead, Microsoft Edge perform OCSP requests for retrieving revocation information regarding each certificate in the chain. OCSP requests are first performed by means of HTTP Gets and since they are not accepted by OCSP responders, new HTTP Posts are automatically generated by the browsers. Microsoft Edge correctly refuses connection with experimental server in presence of a revoked certificate installed in the chain (at any level); it falls back to CRL in case OCSP responders are not available and requests CRL whether receives an *unknown* response status from an OCSP responder. In case both OCSP responder and CRL are not available, Microsoft Edge silently accepts the connection for EV and non-EV certificates.

Safari

Safari, as Microsoft Edge and Chrome for Windows, handles non-EV certificates differently from EV certificates. For non-EV certificates, it does not caches revocation information relate to each certificate in the chain (no OCSP and no request for downloading CRL). For this reason tests show that Safari establish connection with experimental server even in presence of a revoked certificate in the chain installed on the experimental server: it fails on detecting whether a certificate has been revoked or not. For EV-certificates, instead, Safari perform OCSP requests for retrieving revocation information related to intermediateCA and leaf certificates. These requests are malformed for the OCSP responders and Safari does not sends OCSP through HTTP Posts and does not request to download CRLs. Since Safari is not able to cache revocation information, it always accepts connection with experimental server even when a revoked certificate is present in the installed chain. This happens because Safari does not sends HTTP Post and does not fall back to CRL after HTTP Gets fail. Since OCSP requests fail because they are performed through HTTP Gets, it is not possible testing Safari behaviour when receives *unknown* status from OCSP responder.

	Chrome			Firefox	Opera			IE	Edge	Safari
	L	M	W	L/M/W	L	M	W	W	W	M
OCSP Not Available	X*	X	EV	X	X*	X	EV	V L/I	EV	X
CRL Not Available	X*	EV** L/I	EV	L	X*	EV** L/I	EV	V L/I	EV	EV** L/I
Certificate Revoked	X	EV** L/I	EV	L	X	EV** L/I	EV	V L/I	EV	EV** L/I
Fall to CRL	X	X	EV	X	X	X	EV	V L/I	EV	X
Reject unknown status for leaf certificate	X*	EV**	EV	V	X*	EV**	EV	Fall to CRL	EV	EV**
Reject unknown status for int.CA certificate	X*	EV**	EV	X	X*	EV**	EV	Fall to CRL	EV	EV**
Reject unknown status for rootCA certificate	X*	X	EV	X	X*	X	EV	X	EV	X
Reject connection in absence of revocation information	X	X	X	X	X	X	X	X	X	X

Table 5.3: Browser test results. X means browser fails tests in all cases (leaf, intCA and rootCA); V means browser passes tests in all cases; EV means browsers passes test only in presence of EV certificates; X* means browser fails tests in all cases due to it does not perform any kind of requests for retrieving revocation information (no OCSP and no CRL); ** means OCSP requests are performed by means of HTTP Gets and browsers does not perform HTTP Post on HTTP Gets failure: in this case it is not possible to evaluate correctly the behaviour; L means browser passes test only for leaf certificate; L/I means browser passes tests only for intCA and leaf certificate

Chapter 6

TLS implementations

In this chapter I will present some libraries which are able to emulate a TLS client to use for establishing a secure TLS connection with web servers. OpenSSL, GnuTLS and Botan are the only libraries I found which make available a command line utility for the desired purpose.

6.1 Presentation

6.1.1 OpenSSL

OpenSSL [23] is a software library (Written in C, Assembly and Perl) for application which secure communications and contains an implementation of TLS protocol. It supports X.509 certificates handling and validation. It is available for Linux, MacOS and Windows. OpenSSL library uses "verify" command to verify certificates chain, together with some useful options [24].

Verify operation builds up a certificate chain using as starting point the supplied certificate and terminate in the root CA. In case the whole chain cannot be built up, it returns an error. The chain is constructed by looking up the issuers certificate of the current certificate and when a certificate is found which is its own issuer, it is assumed to be the root CA.

"Looking up the issuers certificate" process involves several steps. In OpenSSL versions before the 0.9.5a the first certificate whose subject name matched the issuer of the current certificate was assumed to be the issuers certificate; while in OpenSSL 0.9.6 and later all certificates with a correspondence among their subject name and the issuer name of the current certificate, are subject to additional tests. The relevant authority key identifier components of the current certificate (if present) must match the subject key identifier (if present) and issuer and serial number of the candidate issuer, in addition the keyUsage extension of the candidate issuer (if present) must permit certificate signing.

Firstly, in the lookup procedure, the list of untrusted certificates is explored and if no match is found the remaining lookups are from the trusted certificates. The root CA is always looked up in the trusted certificate list: if the certificate to verify is a root certificate then an exact match must be found in the trusted list.

Verify operation continues checking every untrusted certificate's extension for consistency with the supplied purpose; if the -purpose option is not included, verify does not perform any checks. Leaf certificate extensions must be compatible with the supplied purpose and all other certificates must also be valid CA certificates.

The method checks root CA trust settings, and in particular the root CA should be trusted for supplied purpose (a certificate without any trust settings is considered to be valid for all purpose).

At the end certificate chain validity is checked: for each element in the chain, including the root CA certificate, the validity period as specified by the notBefore and notAfter fields is checked against the current system time. The certificate signature is also checked at this point (except

for the signature of the typically self-signed root CA certificate, which is verified only if the -check_ss_sig option is given).

In case all operations terminate successfully, the certificate in considered valid; otherwise if any operation fails the certificate is target as non valid.

OpenSSL documentation is available at https://www.openssl.org/docs/.

6.1.2 GnuTLS

GnuTLS [25] is a free software implementation (written in C and Assembly) of TLS, SSL and DTLS protocols which support X.509 certificate handling and validation.

It is available for Linux, MacOS and Windows and documentation at https://gnutls.org/manual/gnutls.pdf.

6.1.3 Botan

Botan [26] is a C++ cryptographic library which supports X.509 certificates handling and validation.

It is supported for Linux, MacOS and Windows. Documentation is available at https://botan.randombit.net/handbook/botan.pdf

Table 6.1: Crypto Libraries

OpenSSL
GnuTLS
Botan

6.2 Required command for establishing TLS connection

In this section I will analyse tools presented in 6.1 focusing on the command needed for impersonating a TLS client and simulate a TLS connection with a web server.

6.2.1 Openssl

OpenSSL allows to implement a generic client which connect to a remote host (specified during the call) using TLS. It is possible to specify some options useful for customizing connection. Options I have used are:

- -connect 'host:port': specifies the host and the port to connect to. Default port is 443;
- -servername 'name': set the TLS SNI (Server Name Indication) extension in clientHello message;
- -verify_return_error: option needed for returning verification error;
- -showcerts: option used for displaying server certificates list.

With commands in figure 6.1 I established a connection, using TLS, with google.com, setting all options presented above.

Chain sent by google.com is composed of 3 certificates: for each one openssl performs a verification which, in this case, ends without error (verify return: 1). Having specified -showcerts options, openssl has listed the certificates in PEM format, followed by some information related to TLS connection and the result of chain verification process as showed in 6.2. In this case verification ends without errors.

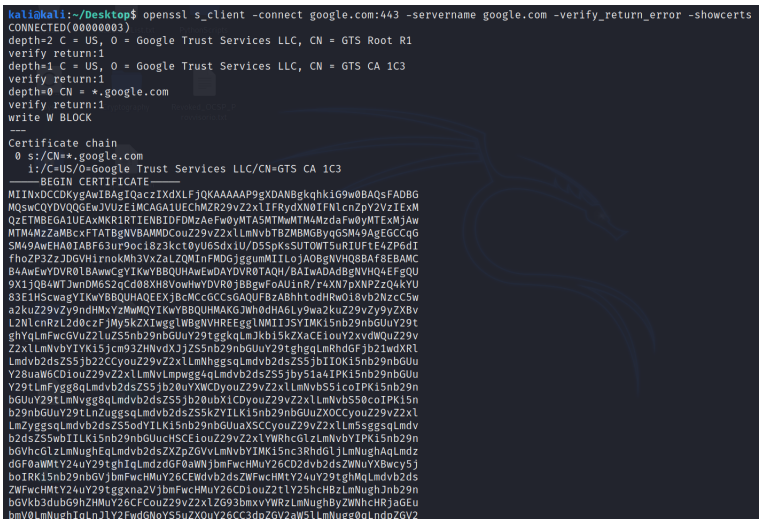


Figure 6.1: Connection with `google.com` over TLS using openssl command

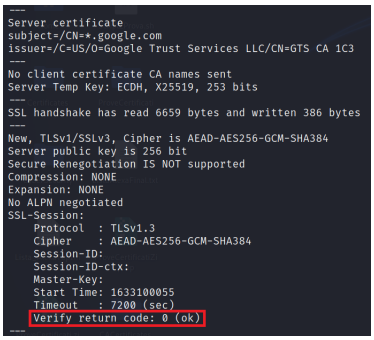


Figure 6.2: Result of connection with `google.com` over TLS using openssl command

Openssl with an expired certificate

In order to test how openssl s_client acts with expired certificates, I used as host `expired.badssl.com` that advertises an expired certificate. Result of connection 6.3 shows that the certificate at depth 0 in the chain has expired and the verify return 0 (error code).

Figure 6.4 shows information related to TLS connection followed by the result of verification over the entire chain: verify returns error code number 10 (certificate expired).

Openssl with a revoked certificate

In order to show how openssl s_client behaves when connecting to a host advertises a revoked certificate, I used as host `revoked.badssl.com`. Surprisingly openssl verify ends without errors because it does not check the certificate status against OCSP or CRL.

In order to check revocation status of `revoked.badssl.com` certificate, I download the certificate and the on of its issuer; then I extracted the OCSP uri from AIA extension and I performed a request to the OCSP responder as showed in 6.6. The certificate has been revoked on 7 October 2019.

OpenSSL s_client emulates a TLS client, requests the server certificate chain, verifies certificates but not their revocation information against OCSP or CRL.

Figure 6.3: Connection with `expired.badssl.com` over TLS using openssl command

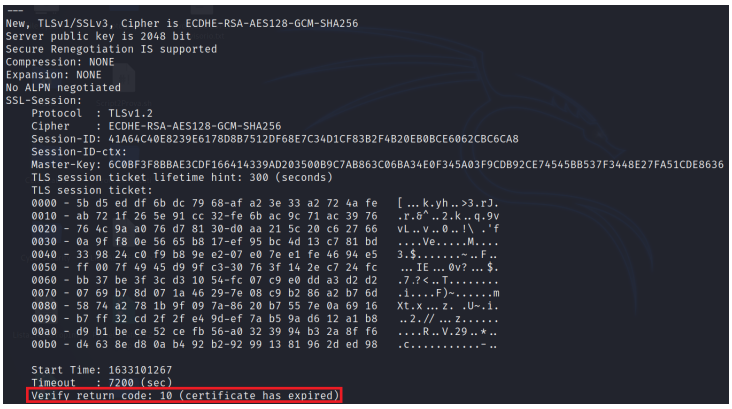


Figure 6.4: Result of connection with `expired.badssl.com` over TLS using openssl command

6.2.2 GnuTLS

GnuTLS is another library that gives the possibility to establish a connection with a server over TLS by means of 'gnutls-cli' command followed by:

- host: specifies the host connect to;
- port: specifies the port (default is 443);
- –print-certs: allows to print certificates sent by the server;
- –ocsp: enable certificate status checking against ocsp.

I established a TLS connection using gnuTLS command with `google.com`. The connection output gives the same information of one obtained with openssl command but it is different in format. As figure 6.7 depicts, connection information are more compacted and contains always result of verification as for openssl s_client: the certificate is trusted.

GnuTLS with an expired certificate

As OpenSSL, gnuTLS notifies an error when try to establish a TLS connection with a web server with an expired certificate.

Figure 6.8 shows that verification fail due to the presence of an expired certificate in the chain. gnuTLS acts as openssl in presence of an expired certificate in the chain.

GnuTLS with a revoked certificate

I tested gnuTLS behaviour when establishes a TLS connection with a web server embedded with a revoked certificate. Without specifying any additional option, gnuTLS does not check certificate status by default.

As picture 6.9 shows, certificate chain verification ends without any error: the certificate is trusted. In order to check whether certificate has been revoked, it is possible to specify –ocsp

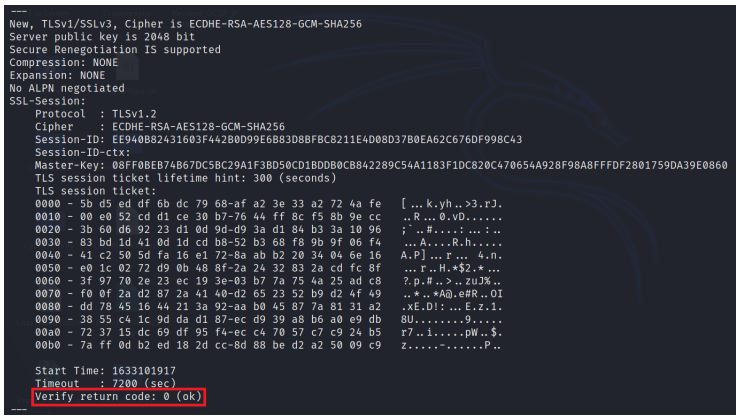


Figure 6.5: Result of connection with revoked.badssl.com over TLS using openssl command

Figure 6.6: OCSP query for checking revocation status of certificate advertised by revoked.badssl.com using openssl command

option that allows to query an OCSP responder for checking certificate status. Establish a TLS connection through gnuTLS library without –ocsp option means not checking certificate status and so establish a connection with a possible malicious web server.

Figure 6.10 shows how the connection output change when connecting always with revoked.badssl.com but specifying –ocsp option: the certificate chain is trusted, but chain verification fail due to presence of a revoked certificate. By comparing openssl and gnuTLS in establishing a TLS connection, it is easier and faster checking certificate status with gnuTLS rather than with openssl because gnuTLS library provides a custom command for this purpose while openssl obliges you to perform manually an OCSP request by means of commands of openssl library.

6.2.3 Botan

Botan library allows to create a connection with a web server over TLS by means of the following commands:

- tls_client: connect to a host using TLS;
- –host: specifies the host connect to;
- –port: specifies the port connect to;
- –print-certs: prints certificate chain.

In figure 6.11 I have established a TLS connection with google.com by means of botan commands. As figure depicts, the library checks certificate status by default against OCSP responder. In this case, TLS handshake ends without any trouble: certificate is trusted (including OCSP check).

Figure 6.7: Connection with google.com over TLS using gnuTLS command

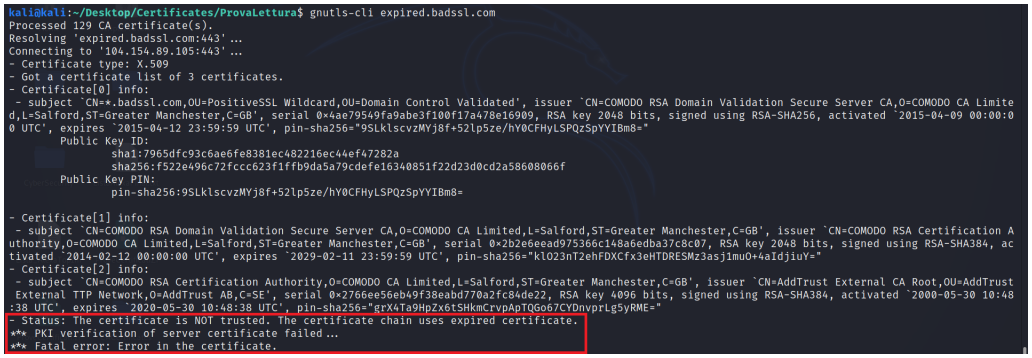


Figure 6.8: Connection with expired.badssl.com over TLS using gnuTLS command

Botan with an expired certificate

In order to test how botan behaves in establishing a TLS connection with a server that advertises an expired certificate, in figure 6.12 I connected with expired.badssl.com that advertises an expired certificate. As expected command return 'certificate has expired' as certificate status.

Botan with a revoked certificate

In order to test how botan behaves in establishing a TLS connection with a server that advertises a revoked certificate, in figure 6.13 I connected with revoked.badssl.com that advertises a revoked certificate. Differently from gnuTLS library and openssl, botan tls_cli command check certificate status against OCSP by default: no additional options or additional commands are needed to check whether the certificate has been revoked.

6.3 Remote Verification

Modern browsers establish secure connections with web server by means of different TLS protocol implementations. Some of these have been presented and discussed in 6.2 and one scope of this study is to compare some different TLS protocol implementations. For this reason I wrote 3 bash scripts for testing OpenSSL, gnuTLS and Botan TLS implementation that establish a secure TLS connection with each one of web server listed in the AlexaTop1M list updated at 26 of August. The scripts check also connection output and will create as many files as different certificates chain verification results present inside the TLS connection output. At the end each file will contain the list of web servers that end certificate validation process with status equal to the name of the file.

Figure 6.9: Connection with revoked.badssl.com over TLS using gnuTLS command

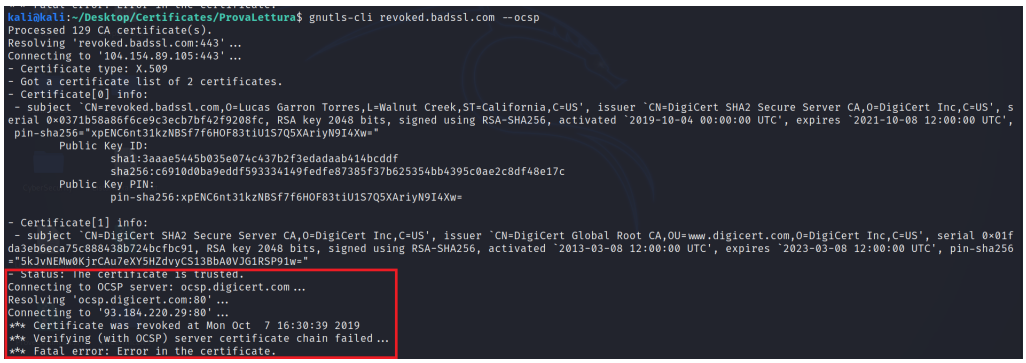


Figure 6.10: Connection with revoked.badssl.com over TLS using gnuTLS command and –ocsp option

6.3.1 OpenSSL

Script Configuration OpenSSL tries to establish a TLS connection with web servers by means of the following bash script:

```
#!/bin/bash
HOST=$1

VER_RESULT= timeout 60s openssl s_client -connect $HOST:443 -servername $HOST
    -verify_return_error </dev/null 2>/dev/null | grep -Eo 'Verify return
    code: [^"]*'

if [[ "$VER_RESULT" == "" ]]; then
        echo "Connection error!"
        echo $HOST>>"Connection_Error_OpenSSL.txt"
else
        echo "$VER_RESULT"
        echo $HOST>>"${VER_RESULT}.txt"
fi
```

Openssl verification results In case of connection error the connection result will be empty, and the url of web server will be written inside the `Connection_Error_OpenSSL.txt` file; otherwise it will be written inside the file with the corresponding name and in case the file is not present yet, it is created.

This script built 13 different output files, listed in table 6.2.

Figure 6.11: Connection with google.com over TLS using botan commands

Figure 6.12: Connection with expired.badssl.com over TLS using botan commands

6.3.2 GnuTLS

Script configuration GnuTLS tries to establish a TLS connection with web servers by means of the following bash script:

```
#!/bin/bash
HOST=$1

VER_RESULT=timeout 60s gnutls-cli $1 </dev/null 2>/dev/null | grep 'Status: '
    | grep -Eo 'The certificate [^"]*'
echo "$VER_RESULT"

if [[ "$VER_RESULT" == "" ]]; then
        echo $HOST>>"Connection_Error_GnuTLS.txt"
elif [[ "$VER_RESULT" == *"The certificate is trusted."* ]]; then
        echo $HOST>>"Verification_OK_GnuTLS.txt"
else
        echo $HOST>>"${VER_RESULT}.txt"
fi
```

GnuTLS verification results In case of connection error the connection result will be empty, and the url of web server will be written inside the `Connection_Error_GnuTLS.txt` file; otherwise if the certificate validation process ends without any error, it will be written into `Verification_OK_GnuTLS.txt`; if validation ends with a security error, url is appended into the file with the corresponding error name and in case the file is not present yet, it is created.

This script generates 33 different output files, listed in tables 6.3 and 6.4.

6.3.3 Botan

Script configuration Botan tries to establish a TLS connection with web servers by means of the following bash script:

```
#!/bin/bash
HOST=$1

VER_RESULT=timeout 60s botan tls_client $1 </dev/null 2>/dev/null | grep
    'Certificate validation status:'
if [[ "$VER_RESULT" == "" ]]; then
        echo "Connection error!"
        echo $HOST>>"Botan_Error_Connection.txt"

else
        echo "$VER_RESULT"
        echo $HOST>>"${VER_RESULT}.txt"
```

Figure 6.13: Connection with `revoked.badssl.com` over TLS using botan commands

OpenSSL		
OpenSSL Verify return code	**Name of file**	**Number of entry**
0	Verify return code: 0 (ok).txt	579.291
1	Verify return code: 1 (error number 1).txt	2
10	Verify return code: 10 (certificate has expired).txt	10.490
13	Verify return code: 13 (format error in certificate's notBefore field).txt	8
14	Verify return code: 14 (format error in certificate's notAfter field).txt	8
17	Verify return code: 17 (out of memory).txt	35
18	Verify return code: 18 (self signed certificate).txt	6875
19	Verify return code: 19 (self signed certificate in certificate chain).txt	98
20	Verify return code: 20 (unable to get local issuer certificate).txt	5708
29	Verify return code: 29 (subject issuer mismatch).txt	3
47	Verify return code: 47 (permitted subtree violation).txt	3
53	Verify return code: 53 (unsupported or invalid name syntax).txt	16
	Connection_Error_OpenSSL.txt	42.805

Table 6.2: Codes returned from certificate validation process during TLS handshake using OpenSSL

```
fi
```

Botan verification results In case of connection error the connection result will be empty, and the url of web server will be written inside the `Botan_Error_Connection.txt` file; otherwise it will be written inside the file with the corresponding name and in case the file is not present yet, it is created.

This script generates 12 different output files, listed in table 6.5.

6.4 Results

Tables 6.2, 6.3, 6.4 and 6.5 show results of certificates validation procedure when a TLS client connects to a web server using OpenSSL, GnuTLS and Botan respectively. I emulate a TLS

connection with sites present in the *Alexa Top 1 Million* list updated at 26th of August and I store the certificates validation results differently for each tool.

OpenSSL verification procedure ends correctly, without any security warning (*Verify return code: 0(ok)*), for the 96,14% of website belonging to *Alexa Top 1 Million* list, while for the 1,74% verification procedure fails due to the presence of an expired certificate in the chain (*Verify Return Codice: 10 (certificate has expired)*). OpenSSL recognizes that the 1,14% of tested web servers host a self signed certificate (*Verify return code: 18(self signed certificate)*), while the 0,94% present an issuer certificate not included inside the OpenSSL trusted root list. OpenSSL does not check revocation status of certificates appearing in the chain.

GnuTLS verification procedure terminates correctly, without any security warning (*The certificate is trusted*), for the 88,18% of website belonging to the *Alexa Top 1 Million* list, while for the 9,92% the name bound inside them does not match the expected one. GnuTLS verification procedure returns more types of error rather than OpenSSL as the number of entries in tables 6.3 and 6.4 show and this because when the validation procedure of a certificates chain fails for several errors, GnuTLS does not stop in reporting the first one but reports the entire set. For this reason there are several entries in tables 6.3 and 6.4 which contain the same error together with other ones. GnuTLS finds that the 1,82% of tested web servers host a chain containing an expired certificate, while 12 chains contain a revoked certificate. Finally, GnuTLS verification procedure returns a security warning when it detects that stapled OCSP response are expired, invalid or required by the certificate.

Botan verification procedure ends correctly, without any security warning (*Verified*), for the 88,15% of website belonging to the *Alexa Top 1 Million* list, while for the 1,23% verification procedure is not able to establish trust since the web site hosts a self-signed certificate. For the 8,19% of web servers, verification procedure returns a security warning since certificate does not match provided name while for the 0,58% of web servers host a chain with an expired certificate. Botan perform revocation checking during the default validation process using OCSP and it find out that: 124 web server host a chain containing a revoked certificate, 12 do not list a way for retrieving revocation data, 4728 list OCSP responder which is not valid yet and 17 return an expired OCSP response.

OpenSSL utility treats as valid certificates where the subject name does not match the provided one and for this reason verification procedure terminates without security warning. Additionally OpenSSL does not take care about revocation status of certificates appearing in the chain while GnuTLS checks only stapled OCSP response (when present). Botan instead checks revocation status for the certificates appearing in the chain by means of OCSP queries in absence of valid stapled OCSP response. GnuTLS utility, instead, is the best in reporting error during validation: in case of validation procedure failure, it reports all errors have caused the failure while OpenSSL and Botan stop in reporting only one error. This is why the script used for launching GnuTLS utility has generated 33 different files against the 13 for OpenSSL and 12 for Botan.

GnuTLS	
Name of file	Number of entry
Certificate chain does not match the intended purpose	1
The certificate chain is revoked	2
The certificate chain is revoked. The certificate chain uses expired certificates. The name in the certificate does not match the expected	3
The certificate chain is revoked. The name in the certificate does not match the expected	4
The certificate chain uses expired certificate	3895
The certificate chain uses expired certificate. The certificate requires the server to include an OCSP status in its response, but the OCSP status is missing	5
The certificate chain uses expired certificate. The name in the certificate does not match the expected	2119
The certificate chain uses insecure algorithm	9
The certificate chain uses insecure algorithm. The certificate chain uses expired certificate	4
The certificate chain uses insecure algorithm. The certificate chain uses expired certificate. The name in the certificate does not match the expected	2
The certificate chain uses insecure algorithm. The name in the certificate does not match the expected	7
The certificate issuer is unknown	5381
The certificate issuer is unknown. The certificate chain does not match the intended purpose. The name in the certificate does not match the expected	1
The certificate issuer is unknown. The certificate chain uses expired certificate.	1716
The certificate issuer is unknown. The certificate chain uses expired certificate. The name in the certificate does not match the expected	2554
The certificate issuer is unknown. The certificate chain uses insecure algorithm	10
The certificate issuer is unknown. The certificate chain uses insecure algorithm. The certificate chain uses expired certificate	43
The certificate issuer is unknown. The certificate chain uses insecure algorithm. The certificate chain uses expired certificate. The name in the certificate does not match the expected	454
The certificate issuer is unknown. The certificate chain uses insecure algorithm. The name in the certificate does not match the expected	193
The certificate issuer is unknown. The certificate requires the server to include an OCSP status in its response, but the OCSP status is missing	3
The certificate issuer is unknown. The name in the certificate does not match the expected	6715
The certificate issuer is unknown. The name in the certificate does not match the expected. The received OCSP status response is invalid.	44
The certificate issuer is unknown. The received OCSP status response is invalid.	22
The certificate requires the server to include an OCSP status in response, but the OCSP status is missing	17

Table 6.3: Part 1: Codes returned from certificate validation process during TLS handshake using GnuTLS

GnuTLS	
Name of file	Number of entry
The name in the certificate does not match the expected	46462
The name in the certificate does not match the expected. The certificate requires to include an OCSP status in response, but the OCSP status is missing.	10
The name in the certificate does not match the expected. The received OCSP status response is invalid	1
The received OCSP status response is invalid	17
The revocation or OCSP data are old and have been superseded	4
The revocation or OCSP data are old and have been superseded. The certificate chain uses expired certificate	3
The certificate is not trusted. The signature in the certificate is invalid	2
The certificate is trusted	520481
Connection error	55149

Table 6.4: Part 2: Codes returned from certificate validation process during TLS handshake using GnuTLS

Botan	
Name of file	Number of entry
Botan_Error_Connection	65466
Cannot establish trust	7159
Certificate does not match provided name	47541
Certificate has expired	3399
Certificate is revoked	124
Certificate issuer not found	5665
No revocation data	12
OCSP cert not listed	12
OCSP not yet valid	4728
OCSP response has expired	17
Unable to find certificate issuing OCSP response	2
Verified	511207

Table 6.5: Codes returned from certificate validation process during TLS handshake using Botan

Chapter 7

Conclusions

Web browsers use digital certificates for identifying web server they are connecting to by building a certificates chain (composed of one ore more CA certificates) rooted by a trusted root CA's certificate. CAs play a crucial role in the Public Key Infrastructure, and one of the goals of this study is to perform a snapshot of X.509 certificates hosted by the most popular web sites.

For this purpose I built a dataset, made up of more than 400.000 certificates, and I extracted some interesting data regarding some fields of certificates. Let's Encrypt [11], the most widespread CA, issued more then 55% of certificates belonging to the dataset and it points out how websites' administrators are not interested in paying someone for receiving a digital certificate but they are satisfied with the CA that offers the service for free. CRL distribution points and authority information access extensions are used for retrieving revocation information, since they host the URLs from which to download the Certificate Revocation Lists and the URLs for contacting OCSP responders. CRL distribution points extension is present only in the 38,42% of end-entity's certificates highlighting how OCSP is the preferred method for checking revocation information appearing in the 99,08% of end-entity certificates. A different situation emerges from the analysis performed on CA's certificates, where the two extensions appear almost in the same percentage of certificates (40,37% CRL and 38,42% OCSP). Only the 1,44% of end-entity certificates are expired while, surprisingly, the 20,67% of collected CA's certificates are expired. OCSP stapling is a standard which let server able to cache OCSP response and send it to clients during TLS handshake as part of *TLS Certificate Status Request* extension. From the analysis performed by Liu et al[3] in 2015, it is clear how only the 2,60% of web servers support OCSP stapling while this study shows that the percentage grow until 45,39% at the end of 2021. Finally, results demonstrate how the 80% of certificates belonging to the leaf set support SCT extension.

Another goal of this study was to check whether web browsers perform correctly certificates validation process, focusing on how they check revocation status of certificates appearing in the chain under several conditions (e.g OCSP responder not available, CRL not available). In case a browsers does not perform correctly validation process, it compromises security of the user and of the information entered. Surprisingly, I discovered some weaknesses in browsers validation process since they behave differently according to the type of certificates (Standard certificate or EV certificate) and the operating system: only Internet Explorer checks revocation status of end-entity and intermediate CA certificates appearing in the chain first performing an OCSP request and falling to CRL in case of failure. Chrome under Windows OS, check revocation status of all certificates appearing in the chain only in presence of EV certificates. The most critical aspect emerged from the study of the web browsers' certificate validation process is the ease with which they establishes a secure connection in the absence of revocation information. This *soft-fail* approach followed by all browsers is threatening for users' experience: it rewards the user's usability but it could has catastrophic consequences on the security of the PKI.

Nowadays there are several crypto-libraries which provide already implemented functions for establishing a secure TLS connection. The final goal of this study was to reveal how 3 different command-line utility libraries (OpenSSL, GnuTLS and Botan) perform certificates validation when they establish a secure TLS connection. The three libraries report different results since

OpenSSL mark as valid the 96,14% of certificates against the 88,18% of GnuTLS and the 88,15% of Botan. The percentage of certificates recognized as valid for Botan is almost the same of GnuTLS, while OpenSSL consider valid some certificates that the other tools mark as invalid.

Bibliography

[1] H. Hoogstraaten, "Black Tulip Report of the investigation into the DigiNotar Certificate Authority breach", 08 2012, DOI 10.13140/2.1.2456.7364

[2] M. M. Berbecaru D., Lioy A., "On the complexity of public-key certificate validation", Information Security. ISC 2001. Lecture Notes in Computer Science, vol. 2200, 2001, pp. 1–1, DOI https://doi.org/10.1007/3-540-45439-X_13

[3] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An end-to-end measurement of certificate revocation in the web's pki", Proceedings of the 2015 Internet Measurement Conference, New York, NY, USA, 2015, pp. 183–196, DOI 10.1145/2815675.2815685

[4] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, J. Rula, N. Sullivan, and C. Wilson, "Is the web ready for ocsp must-staple?", Proceedings of the Internet Measurement Conference 2018, New York, NY, USA, 2018, pp. 105–118, DOI 10.1145/3278532.3278543

[5] D. Berbecaru, A. Lioy, and M. Marian, "Security aspects in standard certificate revocation mechanisms: a case study for ocsp", Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications, 2002, pp. 484–489, DOI 10.1109/ISCC.2002.1021719

[6] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, "Mission accomplished? https security after diginotar", Proceedings of the 2017 Internet Measurement Conference, New York, NY, USA, 2017, p. 325?340, DOI 10.1145/3131365.3131401

[7] D. Berbecaru, "Mbs-ocsp: an ocsp based certificate revocation system for wireless environments", Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004., 2004, pp. 267–272, DOI 10.1109/ISSPIT.2004.1433737

[8] A. S. Wazan, R. Laborde, D. Chadwick, R. Venant, A. Benzekri, E. Billoir, and O. Alfandi, "On the validation of web x.509 certificates by tls interception products", IEEE Transactions on Dependable and Secure Computing, 2020, pp. 1–1, DOI 10.1109/TDSC.2020.3000595

[9] J. M. H. M. e. a. Zulfiqar, M., "Tracking adoption of revocation and cryptographic features in x.509 certificates", Int. J. Inf. Secur., 2021., November 2021, DOI 10.1007/s10207-021-00572-5

[10] R. Roberts, Y. Goldschlag, R. Walter, T. Chung, A. Mislove, and D. Levin, "You are who you appear to be: A longitudinal study of domain impersonation in tls certificates", Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2019, pp. 2489–2504, DOI 10.1145/3319535.3363188

[11] "Let's Encrypt." `https://letsencrypt.org`, Accessed: 2021-11-11

[12] R. S. C. M. S. Chokhani, W. Ford and S. Wu, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework." RFC-3647, 2003, DOI 10.17487/RFC3647

[13] S. Kent, "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management." RFC-1422, 1993, DOI 10.17487/RFC1422

[14] S. F. S. B. R. H. D. Cooper, S. Santesson and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." RFC-5280, May 2008, DOI 10.17487/RFC5280

[15] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and D. C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP." RFC 6960, June 2013, DOI 10.17487/RFC6960

[16] Y. N. Pettersen, "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension." RFC 6961, June 2013, DOI 10.17487/RFC6961

[17] "German researchers obtain a certificate for a domain they do not own." https://www.theregister.com/2018/09/06/certificate_authority_dns_validation/, Accessed: 2021-12-02

[18] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency." RFC 6962, June 2013, DOI 10.17487/RFC6962

[19] "Net Market Share: browser market share." https://netmarketshare.com/browser-market-share.aspx, Accessed: 2021-10-22

[20] "DigiCert Oid." http://oid-info.com/get/2.16.840.1.114412, Accessed: 2021-10-29

[21] "NSS Library." https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS?retiredLocale=it, Accessed: 2021-10-29

[22] "Chromium." https://www.chromium.org/developers/design-documents/network-stack#TOC-SSL-TLS, Accessed: 2021-10-29

[23] OpenSSL Project, https://www.openssl.org/

[24] The OpenSSL project, https://www.openssl.org/docs/man1.1.1/man1/verify.html

[25] GnuTLS, https://www.gnutls.org//

[26] Botan, https://botan.randombit.net/