Ca' Foscari University of Venice

Department of Environmental Sciences, Computer Science and Statistics

Master's Degree in Computer Science

Secure deployment of HTTPS: Analysis and open challenges

Graduand	Lorenzo Soligo
Supervisor	Prof. Riccardo Focardi
Assistant supervisor	Prof. Stefano Calzavara

Academic Year 2019/2020

All things are good
when carried to excess.

Abstract

Users on the Internet unknowingly rely on HTTPS, a protocol whose goal is to cryptographically secure the communication between users and websites by providing confidentiality and integrity. HTTPS relies on the SSL/TLS protocols, but many versions and implementations of these protocols exist and some of them have been proven to be vulnerable to malign attackers. Furthermore, the communication's security depends on other key factors related to a wider application of security best-practices on the web pages: restrictions on the entities that can run code or access cookies, enforcement of the usage of HTTPS, and many more.

In this thesis we analyze the state and security of the HTTPS deployment of the most visited websites for different categories, considering the overall quality of the deployment by evaluating many key aspects.

We carry out an analysis that takes into account the usage of HTTPS itself, the quality of HTTPS certificates, the security of the SSL/TLS implementation used, the presence of server-side cryptographic vulnerabilities, and the adoption of other modern techniques to enforce security.

Finally, we analyze the obtained results and draw some conclusions on the overall state of the HTTPS deployments analyzed. One of the main goals of this work is to raise awareness on the importance of a careful deployment of HTTPS, thus encouraging site operators to keep cryptographic stacks updated and enforce strict security guidelines.

Contents

Chapter 1

Introduction

1.1 Is HTTPS actually secure?

The HyperText Transfer Protocol (HTTP) is the workhorse protocol of the Web, which has been allowing users to surf the Internet for decades. As we will shortly see, however, this protocol does not provide any security guarantee by default. To tackle this fundamental problem, some additional protocols, namely SSL and TLS, have been developed throughout the years with the goal of securing HTTP traffic. When HTTP is run over either of the two, the connection is supposedly *secure*, hence giving birth to HTTPS.

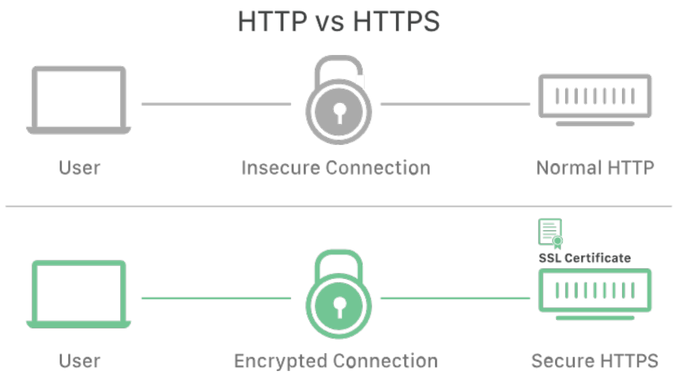


Figure 1.1: HTTP vs. HTTPS, taken from [1]

Excluding SSLv1, which was never published, we can count six versions of these protocols. In chronological order: `SSLv2`, `SSLv3`, `TLS1.0`, `TLS1.1`, `TLS1.2`, `TLS1.3`. Each one of these protocols aims at improving the previous one under different points of view, including security. Indeed, most of them are deprecated as of 2020 [2, 3, 4]. In particular, the French national agency for the security of information systems (ANSSI) has recently published the latest version of its TLS recommendations, officially discouraging the usage of TLS1.0 and TLS1.1 [5]. Many other security concerns arise while deploying HTTPS: for instance, certificates should follow strict best practices, specific HTTP headers should be used to guarantee a higher degree of security, and the usage of cookies should be regulated. In other words, the correct deployment of HTTPS is far from straightforward and requires paying attention to many different key aspects: the adoption of HTTPS is not just a binary checkbox, but rather multiple factors must be taken into account for a realistic security assessment.

This thesis aims at performing a quantitative analysis of the security of each host's HTTPS deployment. The analysis is based on a tool, Discovery[1], on which the author has worked during his internship at Cryptosense. Discovery implements many different security checks on TLS hosts and the logic to compute Attack Trees: both are fundamental for our analysis as they allow us to combine different pieces of information in order to understand whether or not hosts may be vulnerable to attacks. The tool can (and will) be improved further, making it even more complete and fault tolerant.

We hope the results of this work will encourage site operators to take actions to improve the current state of protection of the respective hosts, if needed.

[1] `https://discovery.cryptosense.com`

1.2 Problem statement

In this thesis, we will analyze the quality of the HTTPS deployments of the most visited websites belonging to different categories. The main aspects we will take into account can de divided into different, complementary logical areas.

Adoption and activation of HTTPS at the web application layer

In this thesis we will study the adoption and activation of HTTPS at the *(web) application layer*. The lack of HTTPS support and the use of unsafe practices in the HTTPS activation can entirely void security against network attackers, since communication might run unencrypted. We will thus check whether hosts perform HTTP-to-HTTPS redirects and whether they deploy HSTS, which is a modern technique to enforce the usage of HTTPS for all communication.

Enforcement of the usage of HTTPS

Deploying a website over HTTPS is often not enough to achieve a sufficient level of security: in particular, some resources may still be loaded via plain HTTP and other requests may also be sent unencrypted. This behavior could void security and is thus highly discouraged. In order to enforce the usage of HTTPS, other techniques together with HSTS are available: in particular, Content Security Policy (CSP) is a standard which allows for the usage of two *directives* with the precise goal of avoiding unsafe communication. These directives are called `block-all-mixed-content` and `upgrade-insecure-requests`. We will therefore check how many websites deploy these security mechanisms.

Security of cookies

Strictly related to the state of the HTTPS adoption is the usage of best practices when using cookies. Cookies are small pieces of information that browsers and websites use for many different purposes, including identification: it is therefore imperative to keep them safe. In order to keep them safe, we want the communication to take place via HTTPS and the cookies to be only sent via HTTPS. We will hence study whether the current best practices for cookie security are adopted.

Safe resource inclusion

Loading resources over HTTPS does not necessarily mean that they are safe to be used: in particular, an attacker who has gained access to the host from which resources are included may provide malign files without anyone noticing. In order to be sure that the included resources are actually safe, Subresource Integrity (SRI) can be used. SRI allows for specifying *hashes* of sensitive included resources such as scripts. When a resource is included, its hash is computed; if it does not match with the specified one, the resource is discarded. This makes it basically impossible to load tampered resources. A similar result can be achieved using CSP. We will thus check whether hosts enforce the safe inclusion of resources.

Correct cryptographic implementation of the TLS protocol

Keeping an updated cryptographic protocol is fundamental in order to avoid the possibility of being vulnerable to well-known security problems that affect old deployments. The presence of server-side vulnerabilities, which is usually related to the support of either `SSLv2` or `SSLv3`, represents an urgent problem to be solved as cryptographic flaws in the TLS deployment can reveal cryptographic keys to network attackers, leading to various confidentiality and integrity breaches. SSLv2 and SSLv3 have both been deprecated for years and should absolutely be avoided. We will check for the presence of these well-known vulnerabilities.

Adoption of best practices in HTTPS certificates

HTTPS could not exist without certificates, objects whose goal is to authenticate entities. The adoption of best practices in HTTPS certificates is hence of uttermost importance, as the incorrect management of HTTPS certificates might unduly expose users to phishing attempts or even lead to the disclosure of the cryptographic keys used to protect communication, thus voiding security. We will thus verify the proper usage of HTTPS certificates.

Adoption of modern standards and headers

Although some aspects of HTTPS deployments are not fundamental for security, they are still strong indicators of active maintenance of the deployments and care taken towards their security. We will therefore check whether some websites already support TLS1.3, which has been around for around a year now, and the usage of the `Expect-CT` header, which enforces a quality check on a host's certificate by verifying it against public logs known as *Certificate Transparency* logs.

1.3 Threat model

As per [6], we assume an active network attacker who is able to add, remove or modify messages sent between a client and a server. The attacker also controls a malicious website, which is navigated by the attacked client. By means of the website, the attacker can inject scripts in the client from an attacker-controlled origin, which is relevant for a subset of the considered attacks. However, the attacker can neither break the Same Origin Policy (SOP), nor exploit any bug in the browser. We assume the attacker cannot exploit timing side-channels, since the feasibility of such attacks is generally hard to assess.
In our security analysis, we also occasionally make considerations about passive network attackers, who just sniff the network traffic and do not take actions to avoid detection. These attackers are particularly interesting because they only require very limited skill.

1.4 Thesis outline

This thesis begins with an introduction of all the preliminary concepts required to understand this work. In the next chapters we hence present the theoretical knowledge behind the analysis we performed. Namely, we explain concepts related to HTTP and HTTPS (Chapter 2), moving on to client-side security mechanisms (Chapter 3), HTTPS certificates (Chapter 4), and vulnerabilities at the cryptographic implementation level (Chapter 5). Afterwards, we focus on the main characteristics sought while analyzing a host, describing the setup used to carry out the analysis and the scanning procedure (Chapter 6). Later on, we carry out an analysis of the top websites (according to SimilarWeb[2]) for different categories. In Chapter 7 we present the obtained results and we propose our analysis of them, focusing on what should be improved. In particular, we show that *the current state of many HTTPS deployments is not satisfactory*, as many modern techniques for security such as Subresource Integrity and Content Security Policy aren't nearly as widespread as one might hope. Furthermore, even basic security practices such as the usage of secure cookies are not as commonly used as they should. In our analysis, we also discovered hosts which are vulnerable to attacks on cryptography that have been known for years: these vulnerabilities can have disruptive effects on security and rely on outdated protocols, therefore it is absolutely safe to fix them and they should be mitigated as soon as possible via upgrades. Finally, we explain what could be addressed by future work to obtain more complete analyses and insights. The thesis ends with an overall recap of the work done and our closing remarks.

[2]https://www.similarweb.com/top-websites

Chapter 2

HTTP, HTTPS and TLS

In this chapter we will present the most important concepts needed for understanding how HTTP, HTTPS and TLS work. We will first define three fundamental security properties that we will mention throughout this thesis (Section 2.1). Afterwards, we will present how the HTTP protocol works and its main shortcomings (Section 2.2). Finally, we will explain how TLS can help in securing HTTP and what are some common problems faced while deploying HTTPS (Section 2.3).

2.1 Security properties: definition

Throughout this thesis we will often refer to certain security properties. Their definitions are the following:

- **Confidentiality**: "protecting information from being accessed by unauthorized parties" [7];

- **Integrity**: "ensuring that information is not altered, and that the source of the information is genuine" [7];

- **Authenticity**: "an entity should be correctly identified" [8].

2.2 The HTTP protocol

2.2.1 Main characteristics

The HyperText Transfer Protocol (HTTP) is the most commonly used protocol of the Web. It works at the application layer. Its main usage is the transmission of HTML documents and related resources (media files, ...). Some of its main characteristics are: [9]

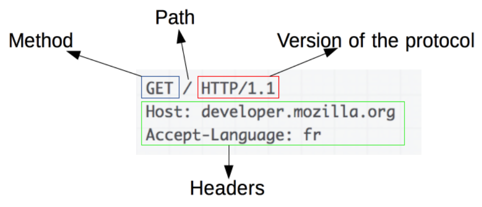
- it is *stateless*. In other words, no state is kept across requests;
- it is a *client-server* protocol. Usually, a web browser communicates with a web server;
- it mostly runs over TCP/IP. It is fundamental that the transport layer protocol is reliable.

2.2.2 Basic functioning

The core functioning of HTTP can be explained as a sequence of *requests* that the client sends to the server and *responses* from the server to the client. Both requests and responses are HTTP *messages* [10]. Messages are composed by standard ASCII plaintext. An HTTP message includes three fundamental components

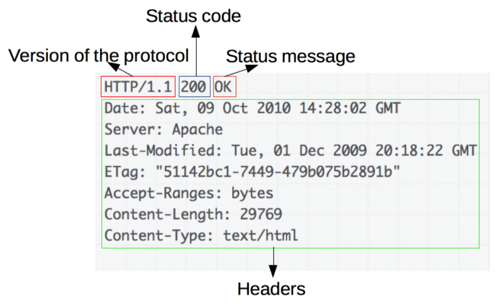
- a *start line* which contains:
  - the request type and endpoint in case of a request
  - the HTTP status code in case of a response
- a set of *headers* (optional)
- a body containing data to be transferred (optional)

In this thesis we will focus on some specific headers, as they can be used to enforce strict security measures if deployed correctly.



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Figure 2.1: An HTTP request, from [11]



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

Figure 2.2: An HTTP response, from [11]

2.2.3 Security concerns

HTTP was designed in the early '90s. As one can imagine, the security of communication was not taken into account at the time. As a matter of fact, HTTP does not provide any confidentiality, integrity or authenticity guarantee by default. Standard HTTP traffic is simple unauthenticated plaintext, which can be read, modified and forged by attackers who are in control of the network, e.g., rogue access points and malicious Internet service providers. TLS aims at solving this problem.

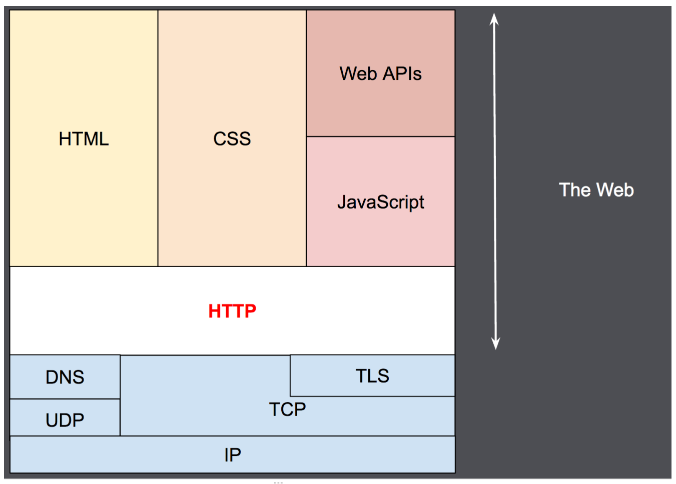


Figure 2.3: Structure of the modern web. Taken from [9]

2.3 Securing HTTP: HTTPS and TLS

2.3.1 Rationale, intuition and limitations

As we have seen, HTTP does not provide any security guarantee by default. This is, of course, not ideal when dealing with sensitive information such as personal data, passwords or credit card numbers.

Luckily, this shortcoming of HTTP can be overcome by the adoption of its secure counterpart HTTPS, which runs HTTP on top of cryptographic protocols like TLS.

HTTPS is an encrypted variant of HTTP based on the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols. HTTPS is designed to guarantee integrity and confidentiality. It also provides authentication using *X.509 certificates*, which we will describe in further detail later on. The idea behind HTTPS is fairly simple: certificates are signed by some trusted parties known as Certification Authorities (CAs), and entities prove to be who they say they are during an initial phase commonly known as TLS handshake (see Section 2.3.4).

HTTPS is phenomenally popular nowadays: as a matter of fact, the amount of HTTPS traffic has recently surpassed the amount of HTTP traffic [12]. Nonetheless, previous research showed that the correct deployment of HTTPS is particularly tricky and things can go wrong at many different levels [13, 14, 15, 16, 17]. As previously said, multiple factors must be taken into account when trying to assess a host's security overall. One of the key takeaways is that the usage of HTTPS alone is a necessary, yet not sufficient condition in order to secure a host, and that the usage of HTTPS itself must be analyzed under many different points of view to assess its quality.

2.3.2 SSL and TLS

In the Introduction, we mentioned that six versions of the SSL and TLS protocols exist, namely `SSLv2`, `SSLv3`, `TLS1.0`, `TLS1.1`, `TLS1.2`, `TLS1.3`.

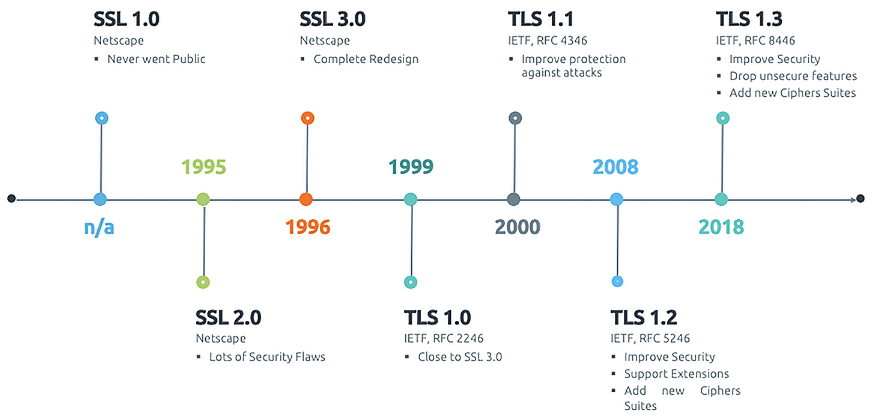


Figure 2.4: Timeline of SSL and TLS, from [18]

Each version improves the previous one under many different key aspects, security included, as per Figure . As of now, only `TLS1.2` and `TLS1.3` are recommended: as a matter of fact, `SSLv2` and `SSLv3` have been known to be vulnerable to many different attacks [14, 13, 17] for years, and have been deprecated since 2011 [2] and 2015 [3] respectively. `TLS1.0` and `TLS1.1` are also being deprecated as of early 2020 [4, 5]. Nevertheless, all these protocols are still being used on many hosts, and although modern browsers now reject connections to hosts using `SSLv2` and `SSLv3`, the confirmed presence of vulnerabilities on such hosts represents a significant security concern.

2.3.3 The TLS protocol: overview

The TLS protocol's most renowned components are the *Record Protocol* and the *Handshake Protocol* [19]. The former is responsible for the negotiation of cryptographic keys and authentication, while the latter is responsible for securing the channel that carries the data. Two other protocols are the *Change Cipher Spec Protocol*, used to assert that the communication starting from a point in time will be encrypted and authenticated, and the *Alert Protocol*, which emits alerts if problems (e.g. in decryption) arise [20].

Before explaining the TLS handshake and cipher suites, it is useful to remind the reader about symmetric and asymmetric key cryptography.
Asymmetric key cryptography relies on a public-private key pair and is usually less efficient than symmetric key cryptography. Symmetric key cryptography, conversely, is usually more efficient. There is only one key, used for both encryption and decryption, that must therefore be kept secret. This key is usually smaller than the ones used in asymmetric key cryptography.
Finally, the concept of *forward secrecy* refers to a scenario in which session keys are not compromised even if the private key of the server is leaked. This is a very desirable condition -for instance, because old sessions could not be decrypted even if a server were to be compromised- that is enforced in TLS1.3.

2.3.4 The TLS handshake

The TLS handshake is implemented slightly differently depending on the specific TLS version. At a high level, however, it can be described as follows [20, 19]:

1. The client initiates a handshake by sending a `ClientHello` message to the server. This proposes a TLS version and a list of supported cipher suites, together with a random *nonce* used for generating keys and an identifier for the session. Some other extensions can be proposed (e.g. for compression, which is usually avoided for security reasons);

2. The server chooses the lower between its highest supported TLS version and the TLS version proposed by the client. It responds with a `ServerHello` message including the chosen TLS version, a random nonce and the session identifier. It then sends a `Certificate` message containing an X.509 certificate to the client. The certificate contains information about the server's identity, the server's public key and the issuing certification authority;

3. The client confirms the validity of (*validates*) the X.509 certificate by checking that it was issued by a trusted certification authority to the hostname it

is trying to connect to, thus getting a proof of authenticity. Further checks may be run, for instance on the revocation status of the certificate;

4. The client and the server take appropriate actions to generate a fresh session key, which is used to protect the communication by means of symmetric encryption, thus ensuring its confidentiality and integrity. Generating the session key may require the server to send a `ServerKeyExchange` message and the client to send a `ClientKeyExchange`. These messages compose the *Premaster Secret* (PMS), which is used with the nonces exchanged in the previous steps to compute the *Master Secret*. Finally, the Master Secret is used to generate session keys for the Record Protocol. At this point, the *Change Cipher Spec* step happens and the subsequent communication will be encrypted and authenticated;

5. The `Finished` message, containing a transcript of the handshake, is exchanged. Different transcripts may indicate some form of tampering during the process.

The session key establishment can be implemented in different ways and takes advantage of the server's public key. The handshake in further detail can be seen in Figure 2.5. As we will see, the way a server manages the TLS handshake is extremely important in assessing its security. As a matter of fact, most of the attacks on cryptography we will see are *padding-oracle attacks* that rely on different answers to different messages from the server.

2.3.5 TLS cipher suites

A TLS cipher suite is a ordered set of cryptographic algorithms. In particular, a cipher suite specifies [22]:

1. the algorithm used for key exchange between the client and the server. It is common to use asymmetric algorithms based on Elliptic Curve cryptography such as ECDHE (Elliptic Curve Diffie Hellman Ephemeral), which also provides *forward secrecy*;

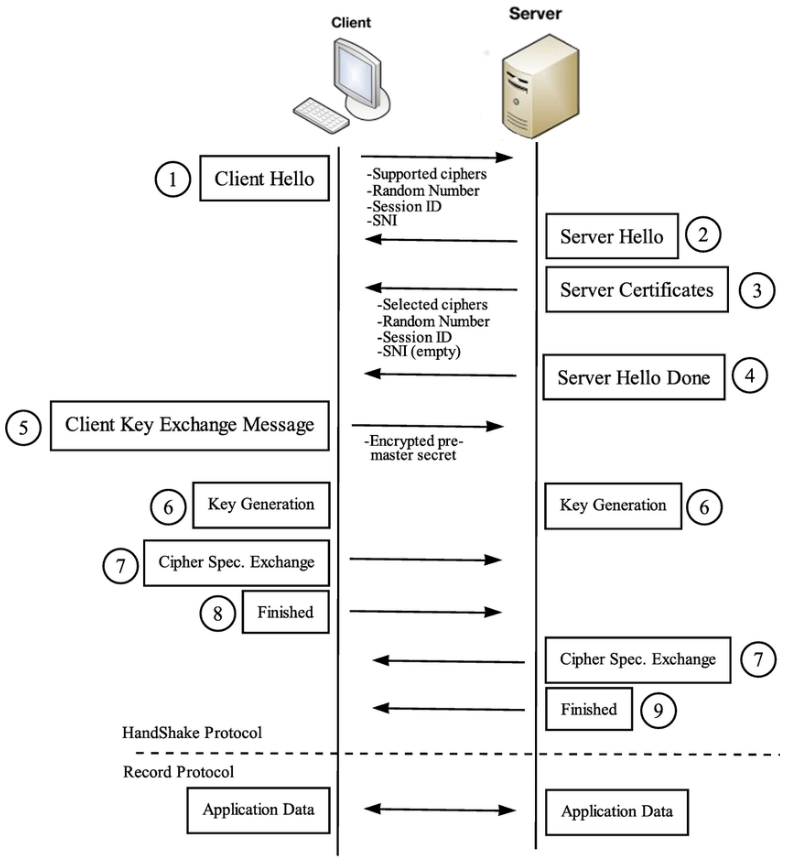


Figure 2.5: TLS handshake, taken from [21]

2. the algorithm used to sign the HTTPS certificate of the server. This algorithm is also asymmetric;

3. the algorithm used for bulk encryption, i.e. to encrypt and decrypt messages between the client and the server. This algorithm is symmetric and needs to provide high performance for large amounts of data;

4. the algorithm used for message authentication: this algorithm's purpose is guaranteeing integrity of the session data through cryptographic signatures and hashes.

For instance, given the cipher suite `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`, we can understand that the key exchange (KE) algorithm is ECDHE, the certifi-

cate signature algorithm is RSA, the bulk encryption is handled using AES_256 in GCM mode and message authentication is obtained via SHA384. We may also find the Elliptic Curve's name after the message authentication algorithm.
It is also important to notice that the latest version of TLS, TLS 1.3, uses the same cipher suite names as the previous versions but with a different meaning, making them non-interchangeable with previous versions. Indeed, TLS 1.3 cipher suites only specify the symmetric cipher for bulk encryption and the message authentication algorithm: for instance, a TLS 1.3 cipher suite would look like `TLS_AES_256_GCM_SHA384`, and as we can see it does not list the key exchange algorithm. In TLS 1.3 all the cipher suites are forward secret and authenticated [23].

Export-grade cipher suites

We refer to export-grade cryptography when talking about encryption whose security is purposedly reduced, mainly by adopting smaller key sizes. The main goal of this process, mostly enforced by the USA, was to limit the availability of non-decryptable cryptography to foreign nations (especially) and the general public [24].

2.3.6 TLS 1.3

TLS 1.2 was introduced in 2008; its successor, TLS 1.3, was published in 2018, ten years and 28 drafts later. Its many advantages include:

- *better security*: as mentioned, TLS 1.3 supports forward secrecy by default. It also disables export-grade and other obsolete ciphers such as MD5, DES, RC4 and CBC-mode ones. It disables RSA key transport too, as it does not provide forward secrecy [25].

- *better performance*: a TLS 1.3 handshake requires one round trip instead of two, hence reducing the time required by the handshake. Also, in case the client has already visited the website, the round trips drop to zero [26].

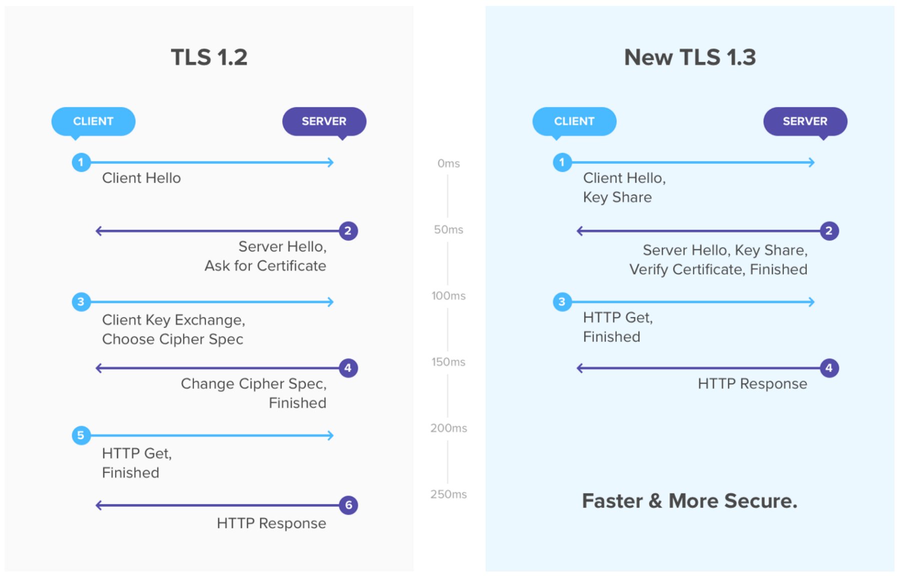

TLS 1.2
CLIENT
SERVER
1 Client Hello
2 Server Hello, Ask for Certificate
3 Client Key Exchange, Choose Cipher Spec
4 Change Cipher Spec, Finished
5 HTTP Get, Finished
6 HTTP Response
0ms
50ms
100ms
150ms
200ms
250ms
New TLS 1.3
CLIENT
SERVER
1 Client Hello, Key Share
2 Server Hello, Key Share, Verify Certificate, Finished
3 HTTP Get, Finished
4 HTTP Response
Faster & More Secure.

Figure 2.6: Comparison between TLS 1.2 and TLS 1.3 handshakes, from [27]

2.3.7 HTTP Strict Transport Security (HSTS)

The `Strict-Transport-Security` header is used to force a browser to access a website only through HTTPS, telling the browser that all HTTP requests should be converted to HTTPS ones [28]. This behavior is strongly encouraged as an attacker may otherwise set up a Man-in-the-Middle attack in case a user connected to the HTTP version of a website, redirecting the user to another website controlled by the attacker. It is also strongly suggested to use the `includeSubDomains` attribute to also force an HTTPS upgrade on subdomains.

It is important to notice that HSTS is a *response* header. This means that by default we must have visited the "benign" website at least once in order for our browser to know that all the connections have to be done through HTTPS. Since this is not always the case, most modern browsers include a list of websites for which HSTS is enabled by default, thus preventing SSL stripping attacks. This list is called *preload list*. Notice, however, that HSTS is orthogonal towards attacks on cryptography such as the ones we will see later on. Finally, it is worth noticing that an ideal HSTS deployment should have the `includeSubDomains` header *and* impose that each subdomain (e.g. https://www.example.com) send a request via HTTPS to the parent domain to enforce the setting of HSTS and protecting the user from possible cookie injection attacks set up by a MITM.

SSL stripping is a man-in-the-middle attack in which an attacker forces a HTTPS to HTTP downgrade in the communication between the client and the server, thus becoming able to steal the user's secrets such as login details, since they are sent as plaintext.

To analyze the practices used to activate HTTPS, we have to check for the presence of redirects. Since redirects can be performed using JavaScript, we cannot just check for the use of `Location` headers and we rather use Puppeteer to access the websites over HTTP using Chromium. We then check the protocol used in the final landing page to see whether HTTPS was activated in some way since the original HTTP request. We also log the presence of HSTS headers in order to understand whether and how it is deployed: in particular, a "basic" HSTS deployment, is not entirely sufficient, because HSTS activation can (and should) be forced on the TLD+1 using the `includeSubDomains` option.
We can, therefore, perform an analysis by placing each host in one of the following categories:

- *No redirection to HTTPS.* This means that the navigation is performed over HTTP by default, which makes network attacks trivial to carry out;
- *Redirection from HTTP to HTTPS, but lack of HSTS adoption.* The website is vulnerable to SSL stripping, yet it is normally served over HTTPS and careful users might notice when the site is unexpectedly served over HTTP;
- *Redirection from HTTP to HTTPS + HSTS.* The website is protected against SSL stripping, but other applications served on sub-domains might be vulnerable to this attack and domain cookies might be leaked or set over HTTP;
- *Redirection from HTTP to HTTPS + HSTS with the* `includeSubDomains` *option on TLD+1.* This ensures that all the applications on the domain are always accessed over HTTPS, as well as granting the confidentiality and integrity of domain cookies.

In case a subdomain's policy depends on an HSTS deployment with the `includeSubDomains` option at a higher level, it should also send a request to that higher-level domain in order to be sure that the `Strict-Transport-Security` header is received and therefore active on the subdomain.

2.3.8 Mixed Content

As we have seen, HTTPS adoption is far from straightforward. Many websites are still not entirely deployed over HTTPS: this means that while some fundamental components of the pages (e.g. the HTML document itself) are loaded via HTTPS, some other "minor" resources such as images may still be loaded using plain HTTP. We define this intermediate condition with the notion of *mixed content*, which we can also divide into two categories: [29]

- *passive* mixed content: resources that are seen as stand-alone, i.e. do not interact with other elements in the same page. In this category we find images, videos, etc.
- *active* mixed content: conversely to its passive counterpart, this is a more sensitive category as these resources interact with the page, resulting in much higher security concerns. The most relevant members of this category are scripts.

While passive mixed content may still be allowed, it is extremely important not to allow any active resource to be loaded over plain HTTP. Anyway, not relying on mixed content is definitely recommended both for security and user experience, since browsers have recently started to block mixed content entirely [30].

Chapter 3

Client-side security mechanisms

In this chapter we will present some fundamental client-side security mechanisms. We will first define Same-Origin Policy (Section 3.1). We will then move on to Subresource Integrity, one of the most efficient ways to safely include resources (Section 3.2). Later on, we will present the concept of cookies and the correct way to manage them in order to enforce the security properties of interest (Section 3.3). Finally, we will explain what Content Security Policy is and how it can help in securing communication (Section 3.4)

3.1 Same-Origin Policy (SOP)

3.1.1 Domains and Sub-Domains

Before talking about Same Origin Policy, it is fundamental to understand the concepts of *domain* and *subdomain*. On the Web, servers are typically identified by a *fully qualified domain name* (FQDN), i.e., a dot-separated sequence of labels terminated by a *top-level domain* (TLD) from a fixed list. For instance, `www.unive.it` is a FQDN under the TLD `.it`. Domain registration typically operates at the granularity of TLD+1: this means that an organization can register a domain name like `unive.it` and then create arbitrary *sub-domains* like `www.unive.it` and `idp.unive.it`. It is common practice to create different sub-domains for different services, e.g., `www.unive.it` to serve the university website and `idp.unive.it` for authentication.

3.1.2 Rationale

Same-origin policy (SOP) is one of the most important and widespread security mechanisms in modern web browsers. It manages how resources loaded from one origin interact with resources from other origins, isolating data and preventing unwanted read/write attempts [31, 32].

3.1.3 Origin: definition

As per [31], we define an origin as a "`(protocol, host, port)` tuple". In other words, two URLs must match in those three fields in order to belong to the same origin. This is a fundamental aspect to understand, as HTTP and HTTPS are two different protocols with two different default ports: the "same" resource -accessed through HTTP and HTTPS- maps to two different origins.

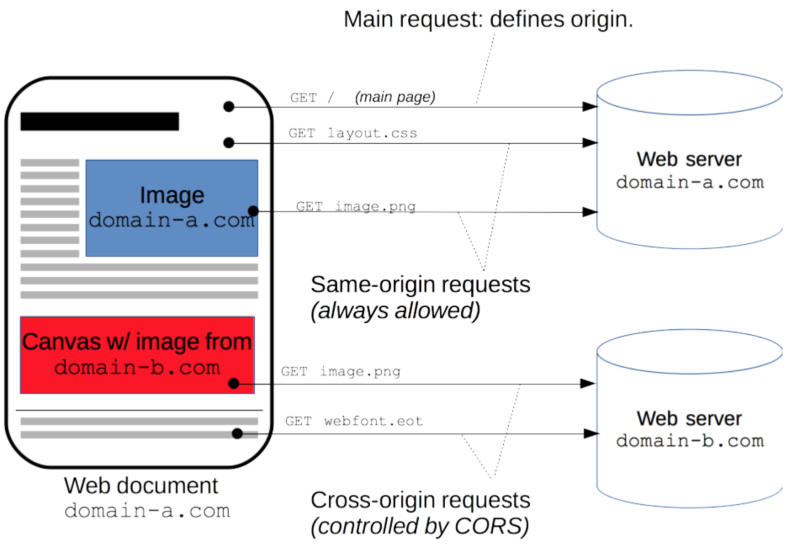


Figure 3.1: Same-Origin and Cross-Origin requests, from [33]

3.1.4 How SOP works

The idea behind SOP is fairly simple: data belonging to each origin must not be read or written by data belonging to other origins. On the other hand, interconnection is one of the cornerstones of the web, and forbidding the inclusion of external content would give the web no sense. Hence, SOP allows for script inclusion using the `<script>` tag, running the included script in the page's origin. Being careful when including external scripts is therefore imperative [32].

3.2 Subresource Integrity (SRI)

3.2.1 Rationale and relationship with SOP

As we have seen, SOP allows for including resources from origins different from the page's. It would thus be useful to have some degree of confidence on the resources being included. In particular, being able to cryptographically verify that those resources have not been tampered with would be a significant guarantee when including external scripts. Subresource Integrity is created with this precise goal in mind.

3.2.2 Definition

Subresource Integrity (SRI) is a recent security mechanism that allows for verifying that content included in the webpage has not been manipulated. This condition is enforced throughout the check of a cryptographic digest stored in the `integrity` HTML tag. If the computed hash is not equal to the one found in the `integrity` attribute, the browser returns a network error and the resource is discarded [34]. SRI is used with `<link>` and `<script>` tags, which import sensitive resources such as CSS and JavaScript files.

3.2.3 Example

SRI requires the developer to specify a base64-encoded hash of a file. The allowed hashes are SHA256, SHA384 and SHA512. An example `integrity` string is the following [34]:

```
<script src="https://example.com/example-framework.js"
integrity="sha384-oqVuAfXRKap7fdgcCY5uykM...">
</script>
```

While parsing the tag, the browser checks that the SHA384 digest of the included resource produces an output which is equal to the given string in base64 representation: if this is not the case, the resource is discarded.

3.3 Cookies

3.3.1 Definition and usage

HTTP cookies, also known as browser cookies, represent data chunks that a server sends and the client stores and sends back in later requests: their main goal is to maintain stateful information.
They serve many purposes, mainly [35]:

- Session management: keeping users logged in, remembering scores in online games, ...
- Personalization: maintaining content personalized, e.g. themes
- Tracking: storing the user's behavior and model the experience throughout analyses of the recorded data

Many attributes can be set in each cookie to achieve the desired goal. Well known goals and attributes are:

- *setting an expiration date and time*, which is obtained using the `Expires` and `Max-Age` attributes

- *defining the cookie's scope*, which is obtained using the `Domain` and `Path` attributes
- *enforcing security*, which is obtained using the `HttpOnly` and `Secure` attributes (flags)

3.3.2 Cookie attributes for security

Usually, the same cookie is shared throughout HTTP and HTTPS connections to the same host. This heavily undermines *confidentiality*, as insecure (HTTP) requests may include cookies set via HTTPS, and *integrity*, since cookies set via the insecure HTTP protocol may be included in requests sent via HTTPS [36].
As previously said, cookies allow for session management and carry personal information regarding the user. Thus, keeping them secure from attackers is of uttermost importance. Many attributes have been introduced with this goal in mind. In particular:

- the `Secure` attribute imposes that a cookie can only be sent through HTTPS and accessed by scripts that run on HTTPS pages [37]. This guarantees confidentiality, but not integrity: indeed, if the same cookies are sent via HTTP, an attacker may be able to modify them, thus undermining integrity. This is known as *Weak Integrity* and modern browsers tackle the issue by not allowing HTTP sites to create `Secure` cookies. Also, a cookie whose prefix is `__Secure-` can be used, as it must have the `Secure` attribute and must be set from an HTTPS URL.
- the `HttpOnly` attribute forbids the browser from exposing the cookie to scripts run on the client. This partially[1] prevents the cookie from possibly being accessed through client-side attacks such as Cross-Site Scripting (XSS), although other techniques such as Cross-Site Request Forgery (CSRF) may still succeed in leaking the cookie.

[1]see [38] for a demo on replacing `HttpOnly` cookies by *overflowing the cookie jar* from JavaScript

- the `SameSite` attribute allows for specifying whether a cookie should or not be sent cross-site. In particular, it is possible to specify that the cookie should be sent: only in same-site requests (`Strict`); in same-site requests and in cross-site top-level requests (`Lax`); in all contexts (`None`, requires `Secure`). This attribute provides protection against CSRF attacks when set in `Strict` mode. [39, 40]

3.3.3 Cookies, SOP and subdomains

The security of sub-domains plays an important role on web application security, most notably because sub-domains can share cookies. For example, `www.unive.it` and `idp.unive.it` can both set cookies with the `Domain` attribute set to `.unive.it`: such cookies, called *domain cookies*, are sent to both services. Though this practice is useful and popular, e.g., to implement authentication across different sub-domains, it also means that attacking `www.unive.it` might break the confidentiality and integrity of cookies at `idp.unive.it` and vice-versa [41]. A possible solution consists of avoiding the usage of the `Domain` attribute altogether, hence allowing the cookie to be only sent to the origin host [42].
As we can see, cookies do not implement SOP as per the specification. In particular, as previously explained, they do not provide isolation by scheme (not by default): HTTP and HTTPS cookies related to the same domain may be shared. The same holds for ports. The previous paragraph also shows how cookies may be shared across different sub-domains (i.e. different origins) thanks to the `Domain` attribute [32].

3.3.4 When is a cookie secure?

As we have seen, correctly identifying secure cookies is a tricky process. Considering as "secure" only cookies that are sent via HTTPS, we can define three conditions for checking whether a cookie is secure or not:

1. the cookie is flagged as `secure`;

2. the cookie is not flagged as `secure`, but it is set on a domain that uses HSTS;

3. the cookie is not flagged as `secure`, but it is set on a domain such that at least one of its ancestors uses HSTS with the `includeSubDomains` option.

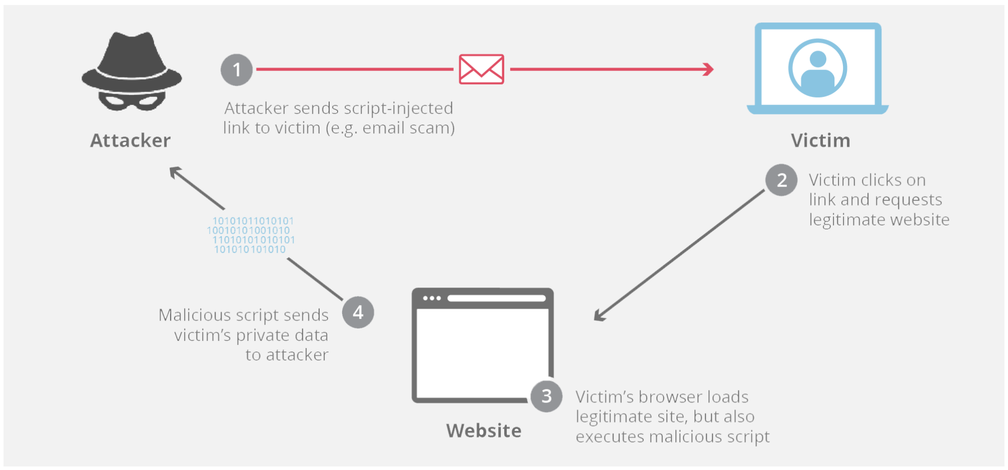
3.4 Content Security Policy (CSP)

3.4.1 Rationale

CSP is a client-side defensive layer that mitigates (and reports) attacks such as XSS. Being client-side, it is enforced by the user's web browser. All the most common web browsers are CSP-compatible. In order to enable CSP, either the `Content-Security-Policy` HTTP response header or the relative `<meta>` tag must be used [43]. As other security mechanisms we have seen, the core idea is to only allow some specific domains to be able to include scripts, hence reducing the attack surface of the webpage. If a script is loaded from a domain that is not whitelisted, the script is not executed. It is also possible to disable the execution of scripts entirely.

3.4.2 Cross-Site Scripting (XSS)

Cross-Site Scripting has steadily been one of the most common attacks on the web for years [44, 45]. It is a quite nasty attack, as it bypasses SOP by running malicious code in the target's origin [46]. XSS usually happens when user input is not properly sanitized or escaped, and can therefore be run in the page. At that point, an attacker may send a malicious URL to the victim, that would leak sensitive data just by clicking on it without even realizing something wrong has happened [47].



Attacker
1
Attacker sends script-injected link to victim (e.g. email scam)
Victim
2
Victim clicks on link and requests legitimate website
Website
3
Victim's browser loads legitimate site, but also executes malicious script
4
Malicious script sends victim's private data to attacker

Figure 3.2: A classic XSS attack, taken from [47]

XSS is one of the reasons why having `HttpOnly` cookies is important: if protects them from possibly being leaked from such an attack.

3.4.3 Directives

CSP relies on its own small declarative language. In a page, a developer would usually configure at least a `default-src` fallback directive, a `script-src` directive for scripts, and a `report-uri` directive for reporting attempted violations of the policy. Other important directives are `style-src` for stylesheets and `connect-src` for the targets of `XMLHttpRequests`.

3.4.4 Enforcing security

Whenever either `script-src` or `default-src` is defined, CSP enforces other restrictions: it forbids the execution of inline scripts, inline event handlers and `javascript:` URLs. It also blocks the execution of `eval()` and limits other functions such as `setTimeout` [48]. Hence, if we want to deploy CSP, we should not use any of these in our code. Since inline scripts may often be useful, we can whitelist them using a random *nonce* for each incoming request, which helps in protecting from script injection. However, this implementation is susceptible to "recursive" script inclusion and does not provide any guarantee on which script is

actually being run. It is usually safer (but also more complex) to whitelist scripts based on their *hashes*, following SRI's principle. This also holds for inline scripts, whose hash is automatically computed.
Finally, CSP can also be used together with many of the mechanisms presented so far (HSTS, SOP, SRI, ...) to enforce the exclusive usage of HTTPS. In particular, two headers can be used to achieve the goal:

- the `upgrade-insecure-requests` header updates all HTTP requests to HTTPS before sending them. This directive is especially useful when migrating from HTTP to HTTPS, as it takes care of upgrading all the possibly insecure requests. In case a resource is not available via HTTPS, the browser avoids loading it overall to enforce security;

- the `block-all-mixed-content` header forbids the usage of any type of mixed content in the page [49]. This directive is also propagated into `<iframe>` resources, with the desirable side effect of the page not loading mixed content at all levels [29].

An interesting property is that `upgrade-insecure-requests` is evaluated before `block-all-mixed-content`. Since the first one already avoids possibly loading resources via HTTP, the second directive does not serve any use if combined with the former [43].
It is easy to notice that correctly configuring CSP is far from straightforward and existing research [50] shows that many websites deploy it in a way that basically voids its most promising security guarantees.

Chapter 4

HTTPS certificates

In this chapter we will explain what X.509 certificates are and their fundamental role in securing HTTPS communication. We will define the key properties of X.509 certificates (Section 4.1) and explain how they are validated (Section 4.2), also presenting the different forms in which certificates come (Section 4.2.2) and the concept of Certificate Transparency (Section 4.2.3).

4.1 X.509 certificates

4.1.1 Public key infrastructure

Understanding public key infrastructures (PKIs) is fundamental in order to understand the rationale and the behavior of X.509 certificates. PKIs can be seen as a set of tools (both hardware and software) and processes which are needed for creating, managing, using and revoking certificates and public keys. We could say their main goal is establishing identities of entities such as people and services [51]. In layman terms, a PKI is something we rely on to authenticate users and their devices. The core idea is that a trusted entity can certify that a cryptographic key belongs to user u, then that key can identify u.

4.1.2 Introduction to certificates

Certificates used in SSL/TLS follow the X.509 standard. Each certificate includes many different fields. Some of the most notable ones are the validity period, the subject (identity) of the certificate and the public key. A certificate may be self-signed, but certificates are usually considered "good" if they are signed by Certificate Authorities (CAs), trusted entities that issue certificates and guarantee the identity of users. A certificate also contains a serial number, which is useful for revocation, and a signature of its body with private key of the issuer (the CA in most cases). Certificates are commonly used in HTTPS communication and email signing and encryption [52].

4.1.3 Certificate policies

Certificate policies are "fields" inside a certificate that describe its properties. Generally speaking, certificate policies are documents whose goal is to describe the roles of different entities in public-key infrastructures. When a X.509 certificate is issued, it is possible to specify in which cases it can (and can not) be used. Some fields are mandatory: for instance, each certificate must be identifiable via its Serial Number. All these properties of certificates are expressed in the form of extension policies, characterized by OIDs (object identifiers).

4.2 Certificate validation

4.2.1 Steps

Certificate validation is usually performed using already existing APIs provided by common tools in the industry such as OpenSSL.
The process can be summarized in three steps [53].

Step 1: build the certificate chain and validate signatures

Before trusting the contents of a certificate we want to verify the certificate's signature. This requires locating the authority that signed the certificate: this is achieved by traversing the intermediate certificates and certificate store looking for a certificate whose *subject* is the *issuer* of the certificate to be validated. In case of multiple matches for the subject alone, it is also possible to match the *Subject Key Identifier* and *Issuer Key Identifier* extensions. In case there are still multiple matches, the most recently issued certificate is generally used. It is now possible to check the signature on the target certificate using the authority certificate's public key. If this step fails, the validation process can already be stopped, with the target certificate considered invalid. Building the certificate chain requires the validator to have all the certificates in the chain; the Server Certificate Validation Protocol (SCVP) is more commonly used for requesting a certificate chain from a server.

Step 2: check validity dates, policy and key usage

After having verified the certificate's signature it is fundamental to check different fields to see if it is currently valid. Each certificate's [`valid not before`, `valid not after`] range must contain the current date. Some X.509 extension fields must also be verified.
In particular:

- the `BasicConstraint` extension is required for CAs and limits the depth of the chains starting from specific certificate;
- the `NameConstraints` extension limits the namespace of identities certified underneath the given CA certificate;
- the `KeyUsage` and `ExtendedKeyUsage` excensions set limits on what a certified key can be used for.

Step 3: consult revocation authorities

Now that the certificate chain is known to have valid dates and correct extensions usage, it is a good practice to check whether a certificate has been revocated by contacting the revocation authorities specified in each certificate. Some extensions in the certificates may contain extensions that point to Certificate Revocation Lists or to Online Certificate Status Protocol (OCSP) responders.

4.2.2 Security practices

It is well-known that certificates should only be signed by a trusted certification authority and should only be considered valid up to a given expiration date. What is likely less known is that certificates come in different forms. In particular, by increasing level of security guarantees:

1. *Domain Validated* (DV) certificates are issued after proving some form of control over a given domain name, but do not provide any form of binding between the domain name and the organization which claims ownership of the domain;
   - we can recognize Domain Validated certificates as they contain the `2.23.140.1.2.1` policy identifier [54].
2. *Organization Validated* (OV) certificates are only issued after proving that a domain name is actually controlled by a given physical organization. This requires the presentation of appropriate documentation about the organization asking for the certificate;
   - these certificates lack fields such as `Business Category`;
   - we can recognize Organization Validated certificates with the `2.23.140.1.2.2` policy identifier [54].
3. *Extended Validated* (EV) certificates are similar to OV certificates, but are subject to even stricter security checks. Browsers often rely on custom security indicators for EV certificates and show the name of the owning organization directly in the address bar.

- these certificates are characterized by some fields in the *Certificate Policies* extension field.
  In particular, the fields `jurisdictionOfIncorporationCountryName`, `businessCategory` and `serialNumber` are mandatory,
  while `jurisdictionOfIncorporationStateOrProvinceName` and `jurisdictionLocalityName` are optional [54].

Major organizations like universities or banks should only use OV or EV certificates, since DV certificates provide no protection against phishing attempts. For example, an attacker could get a valid DV certificate for `www.unvie.it` and host a website which pretends to be the legitimate website of the Ca' Foscari University of Venice (`www.unive.it`).
Moreover, security-conscious administrators should avoid the use of *wildcard* certificates. Wildcard certificates apply to arbitrary sub-domains like `*.unive.it`, hence are typically reused on a multitude of different hosts. This simplifies the HTTPS deployment, but also implies that all such hosts have access to the same cryptographic keys, hence the compromise of any host would suffice to get read and write access to all the HTTPS traffic exchanged with any sub-domain of `unive.it`. Wildcards cannot be used in EV certificates, but it is worth noticing that even certificates which do not make use of wildcards might be unduly issued for a large number of domains by specifying multiple Subject Alternative Names (SANs) in them.

4.2.3 Certificate Transparency (CT)

Rationale

Certificate Transparency is an open source standard for assessing the security of certificates. It consists of an append-only public log of certificates released by CAs. This allows for checking if and what certificates have been issued for a domain name, and alerts a domain's owner if another certificate for that domain name is issued. The `Expect-CT` HTTP header forces a browser to verify that the website's certificate is in the Certificate Transparency logs [55, 49]. Certificate

Transparency introduces three new components in the current HTTPS certificate system: certificate logs, certificate monitors and certificate auditors [56].

Certificate logs

Certificate logs are services that keep records of SSL certificates. They are append-only, cryptographically assured using Merkle Tree Hashes, and publicly auditable, meaning that any entity can query a log to verify that a certificate has been appended.
Although any entity can submit a certificate to a log, most certificates are submitted by certificate authorities and server operators. The log sends a Signed Certificate Timestamp (SCT) -a simple promise to add the certificate to the log within a certain time frame (MMD, Maximum Merge Delay)- in response. A TLS server must deliver the SCT with the certificate during the TLS handshake.

Delivering an SCT with a certificate

There are three methods for delivering an SCT with a certificate, namely [56]:

- `X.509v3 extension`: the CA submits a precertificate to the log, who returns an SCT. The CA then attaches the SCT to the precertificate as an X.509v3 extension, signs the certificate, and delivers the certificate to the server operator;

- `TLS extension`: a special TLS extension can be used by server operators to deliver SCTs. The CA issues the certificate to the server operator, who submits the certificate to the log. The log sends the SCT to the server operator, who uses the `signed_certificate_timestamp` TLS extension to deliver the SCT;

- `OCSP stapling`: SCTs can be delivered by server operators via the Online Certificate Status Protocol (OCSP) stapling. Here the CA issues the certificate to the log server and the server operator at the same time. The server operator then makes an OCSP query to the CA, that responds with the SCT. The server then includes the SCT in an OCSP extension during the TLS handshake.

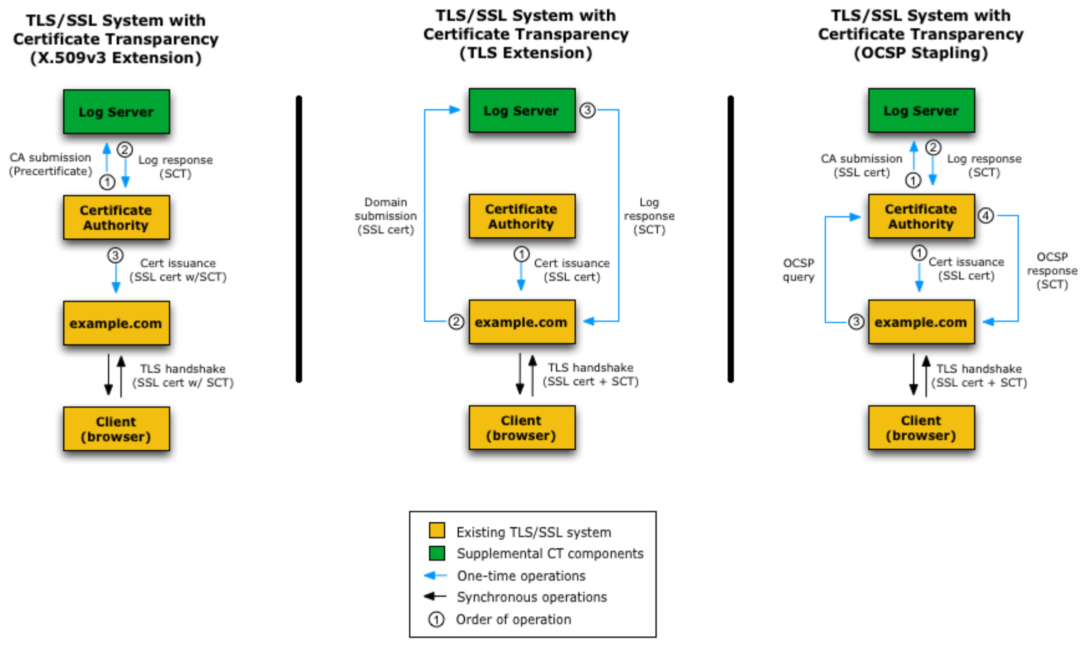


Figure 4.1: Certificate issuance with CT, from [57].

Monitors and auditors

A monitor's goal is to look for suspicious certificates in the logs, for instance illegitimate ones, and to verify that all the logged certificates are visible in the log. Monitors usually keep complete copies of the logs they monitor. Monitors can even behave as backup, read-only logs while logs are offline.
Auditors, on the other hand, verify the overall integrity of logs using log proofs – signed cryptographic hashes that certify a log is reliable. Every log must provide its proofs if asked. Log proofs also allow for checking whether a given certificate appears in a log. Finally, auditors can use log proofs to check that new entries have been added to the log and the log has not been corrupted.

Chapter 5

Cryptographic implementation

In this chapter we will present how (server-side) vulnerabilities at the cryptographic implementation level (SSL/TLS) can have disruptive effects on the security of HTTPS deployments. We will present protocol version downgrade attacks (Section 5.1.1), the highly renowned Bleichenbacher padding-oracle attack (Section 5.1.2), and three other famous attacks: DROWN (5.1.3), ROBOT (Section 5.1.4) and Heartbleed (Section 5.1.5). Afterwards, we will define insecure channels (Section 5.2) and present attack trees, which are the tool we used to perform inferences and understand whether hosts may be subject to attacks (Section 5.3).

5.1 Attacks on TLS

5.1.1 Protocol version downgrade attacks

As we have seen in Section 2.3.4, a TLS server should respond to a `ClientHello` message with either the proposed version of the protocol, or with its highest supported version (if lower than the proposed one). For instance, if a client proposed `TLS1.2` but the server only supported `TLS1.1`, it would respond with `TLS1.1`. Conversely, if a client proposed `TLS1.2` and the server supported all TLS versions up to `TLS1.3`, it would respond with `TLS1.2`. Since some web servers simply drop connections if there is no match in the TLS versions, browser might repeat the handshake procedure with lower protocol versions. An active attacker could, therefore, drop messages to downgrade the communication to vulnerable protocol versions.

POODLE

POODLE (Padding Oracle On Downgraded Legacy Encryption) is a vulnerability found in servers that support SSLv3 with cipher-block chaining (CBC) mode ciphers. In POODLE, an attacker which has already successfully set up a Man-in-the-middle attack forces connection failures to downgrade the communication to SSLv3 and then decrypt part of the SSL communication [59]. While the vulnerability in SSLv3 itself is not fixable, it is strongly suggested to avoid using SSLv3 entirely. A fix for the broken downgrade procedure, conversely, can be found in the `TLS_FALLBACK_SCSV` protocol extension [17]. This consists of a fictitious ciphersuite appended to handshake attempts subsequent to the first one [60]. If this ciphersuite is present and the TLS version in the message is lower than the highest supported from the server, then there may have been an attempt of attack. The great advantage in this extension is that, while some old servers may crash when receiving unknown or unsupported TLS versions in the `ClientHello` message, the extension does not cause any fails.

5.1.2 Bleichenbacher 1998: let padding oracles in

In 1998 Bleichenbacher, whom at the time worked at Bell Labs, noticed the following. Let be given: an RSA public key with base n and exponent e; the relative private key d; an oracle that discerns whether any ciphertext c, once decrypted as $c^d \bmod n$, is correctly padded as per the PKCS#1 standard. Then, this oracle can be used to decrypt or sign a message using an adaptive chosen-ciphertext attack [61]. In the more than twenty years that have passed since Bleichenbacher's discovery, many other attacks have been developed, often relying on the same *padding oracle* concept. Some well-known examples are the OpenSSL CBC padding oracle CVE-2016-2107 [62], POODLE [17], DROWN [13], and ROBOT [14].
More details on the exploitation of the attacks presented in the following sections will be given in the Attack Trees section (Sec. 5.3). We will now quickly present

three famous attacks, of which two are variants of Bleichenbacher's, in order to give the reader some insight on how they work; see the original papers for more information.
Finally, it is essential to understand that if a server is or was vulnerable to such attacks, changing keys and certificate and revocating the old certificate would go a long way in ensuring users' security.

5.1.3 DROWN: Decrypting RSA with Obsolete and Weakened eNcryption

Attack intuition and description

DROWN [13] is a cross-protocol attack that exploits SSLv2 in order to decrypt TLS connections. DROWNS allows for breaking a single host which supports both SSLv2 and TLS by exploiting vulnerabilities in SSLv2 and, in the *special* case, bugs in OpenSSL. The most interesting aspect of DROWN, however, is that it allows the attacker to break a perfectly configured host by means of another, vulnerable host which uses the same RSA public key in its certificate and supports SSLv2.

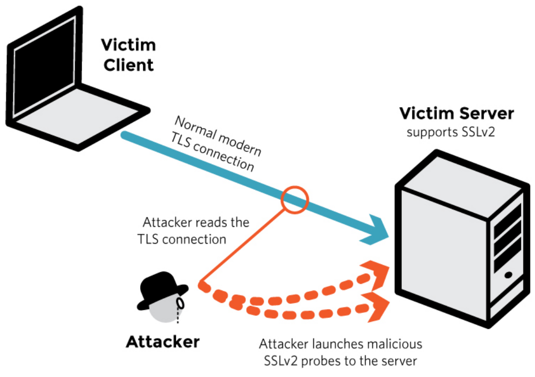

Victim Client
Normal modern TLS connection
Victim Server
supports SSLv2
Attacker reads the TLS connection
Attacker
Attacker launches malicious SSLv2 probes to the server

Figure 5.1: DROWN on a single host. Taken from [13]

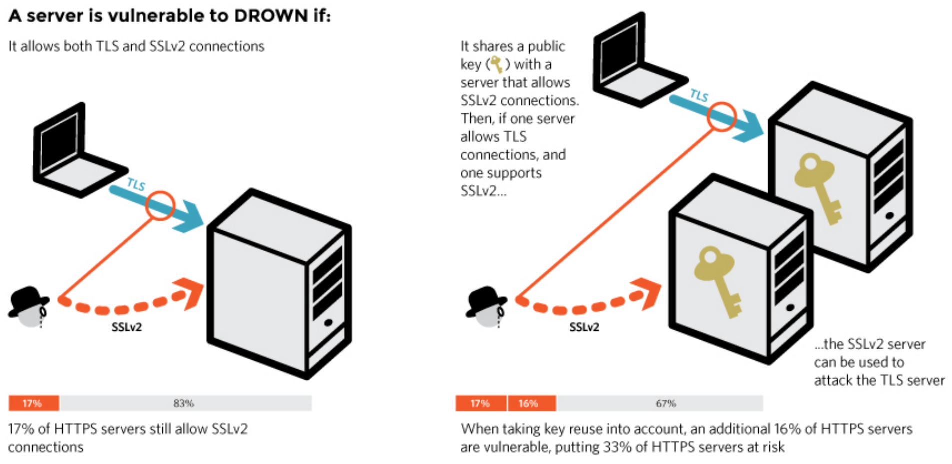

A server is vulnerable to DROWN if:
It allows both TLS and SSLv2 connections
TLS
SSLv2
17%
83%
17% of HTTPS servers still allow SSLv2 connections
It shares a public key with a server that allows SSLv2 connections. Then, if one server allows TLS connections, and one supports SSLv2...
TLS
SSLv2
...the SSLv2 server can be used to attack the TLS server
17%
16%
67%
When taking key reuse into account, an additional 16% of HTTPS servers are vulnerable, putting 33% of HTTPS servers at risk

Figure 5.2: DROWN on two hosts sharing the same key. Taken from [13]

Attacker capabilities

Two variants of DROWN are defined: *general* and *special*. The general variant can be used with any SSLv2 implementation, with the only requirement that it must accept *export-grade* cipher suites. The special variant relies on the `Extra Clear` and `Leaky Export` OpenSSL bugs (with the latter relying, again, on information leaks due to export ciphers) and allows the attacker to break TLS ciphertext in less than a minute on a standard consumer PC, allowing for a Man-In-The-Middle attack.

5.1.4 ROBOT: the Return Of Bleichenbacher's Oracle Threat

Attack intuition and description

As the name of the attack suggests, ROBOT is an updated variant of the attack originally developed by Bleichenbacher in 1998 rather than a completely new attack.

In 2017, Böck et al. [14] realized that the vulnerabilities reported by Bleichenbacher were still present in most major TLS implementations. The test sends five differently formatted PKCS #1 v1.5 messages as `ClientKeyExchange` [63]:

1. Correctly formatted TLS message: `0x00` at the correct position, correct TLS version in the premaster secret. `M1 = 0x0002 || pad() || 0x00 || version || rnd[46]`

2. Incorrect PKCS #1 v1.5 padding: the first byte in the padding is invalid. `M2 = 0x4117 || pad()`

3. `0x00` at the wrong position: results in wrong length in the unpadded premaster secret. `M3 = 0x0002 || pad() || 0x0011`

4. No `0x00` after the padding. `M4 = 0x0002 || pad()`

5. Wrong TLS version. `M5 = 0x0002 || pad() || 0x00 || 0x0202 || rnd[46]`. As a matter of fact, SSL/TLS versions in hex are described as follows: `TLS 1.2=0x303, TLS1.1=0x302, TLS1.0=0x301, SSLv3=0x300` [64]

The only way for a server not to be vulnerable to a chosen-ciphertext attack is to respond with the same alert message to all the five presented messages. If any response is different from the others, then it is possible to perform a Bleichenbacher attack on the server. We can also discern between *strong* and *weak* oracles, with the oracle being strong if and only if we get at least two different responses from messages `M2`, `M3` and `M4`.

Attacker capabilities

As shown in [20], an attacker who is able to exploit a Strong Bleichenbacher oracle on a server could be able to learn the session keys, which allows for decryption, and may even be able to set up a Man-In-The-Middle attack, which also allows for modification of the messages.

5.1.5 Heartbleed

Rationale

Heartbleed is a somehow different attack from the other ones mentioned in this chapter as it is specific to some versions of OpenSSL. In particular, Heartbleed allows the attacker to leak the server's memory, including the keys used for certificates [65].

The core idea behind Heartbleed is to exploit vulnerable OpenSSL versions that do not perform a fundamental sanity check on *heartbeat* requests.[1] In particular, the response's body size is only dependent on the *length* parameter in the request, regardless of the real size of the request's payload. This allows attackers to leak data in memory by simply sending small payloads and asking for much more data.

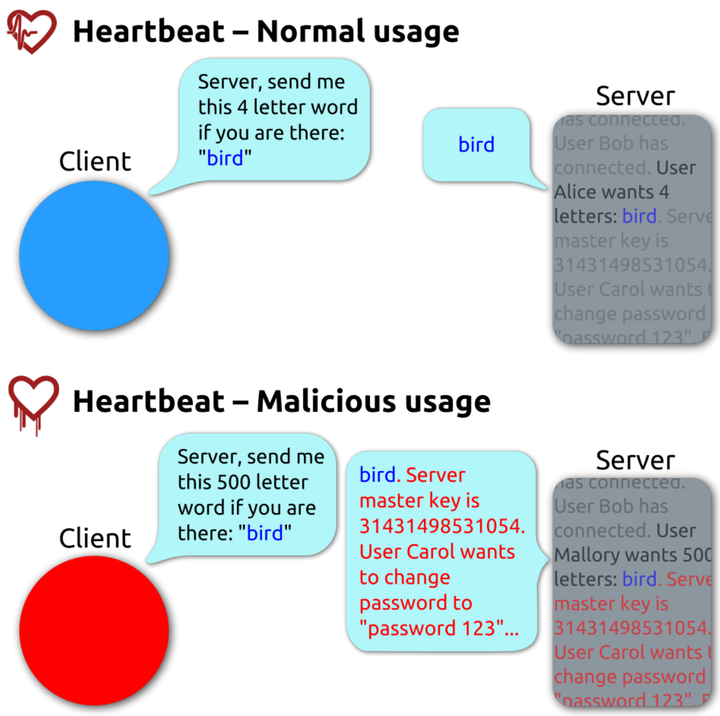

Heartbeat – Normal usage
Client
Server, send me this 4 letter word if you are there: "bird"
bird
Server
User Bob has connected. User Alice wants 4 letters: bird. Server master key is 31431498531054. User Carol wants to change password "password 123".
Heartbeat – Malicious usage
Client
Server, send me this 500 letter word if you are there: "bird"
bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"...
Server
User Bob has connected. User Mallory wants 500 letters: bird. Server master key is 31431498531054. User Carol wants to change password "password 123".

Figure 5.3: An exemplified Heartbleed attack, from [67].

[1]Heartbeat requests are simple requests in which one entity asks the other entity to echo back some bytes as a keep-alive feature [66].

Attacker capabilities

As mentioned, an attacker who is able to exploit Heartbleed on a server could be able to steal keys, certificates, and even dump unrelated data such as instant messages, documents, and emails [65]. Extracting a server's private key using Heartbleed is surprisingly simple, as the attack's implementation is open-source[2].

5.2 Insecure channels

5.2.1 Taxonomy

We define insecure channels as follows [6]:

- *Leaky channels*: established with servers vulnerable to confidentiality attacks, which give the attacker the ability to decrypt the network traffic;
- *Tainted channels*: susceptible to man-in-the-middle attacks, which give the attacker the ability to decrypt and arbitrarily modify the network traffic. Observe that tainted channels are also leaky by definition;
- *Partially leaky channels*: suffering from side-channels, which give the attacker the ability to disclose selected "small" secrets (like cookies) over time. Observe that leaky and tainted channels also qualify as partially leaky.

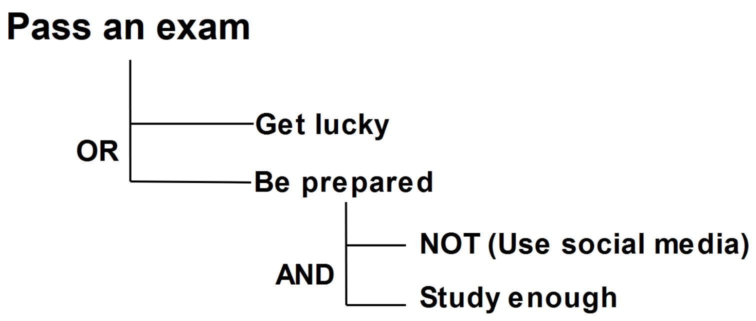
5.3 Attack trees

5.3.1 Definition and rationale

In Section 5.2 we introduced the notions of *(partially) leaky* and *tainted* channels. In order to evaluate these properties, we rely on the Attack Trees defined in [20]. Attack trees are simple diagrams which evaluate whether an entity is vulnerable to a certain attack by returning a Boolean true/false value. These trees are often a few levels deep and can be used as nodes in more complex trees. Nodes are combined

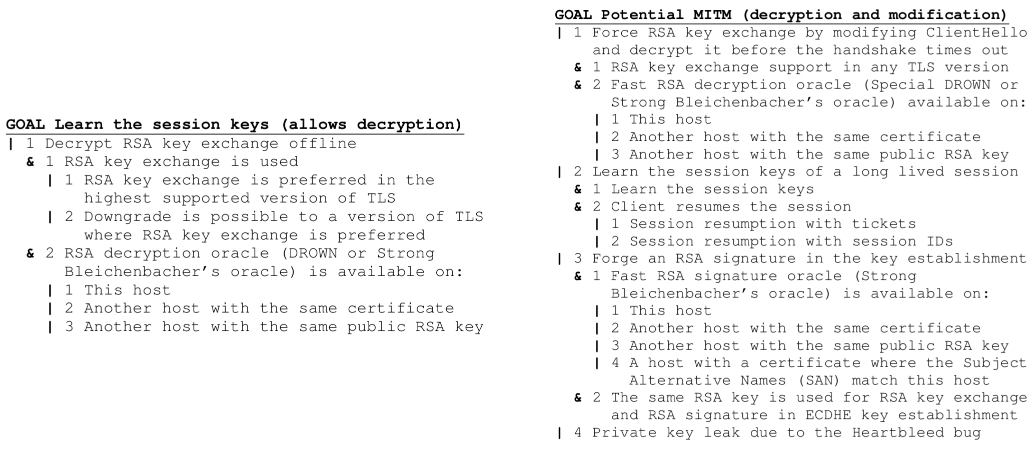
[2]`https://github.com/indutny/heartbleed`

using either the logical conjunction AND or the logical disjunction OR, and can be negated using the NOT operator. Using attack trees is a very suitable way to represent possible attacks on hosts, as attacks frequently require many different conditions to be present at once and are often feasible via different combination of these conditions.



5.3.2 Our implementation

In our analysis, we will check for possible *leaky* and *tainted* channels. The two "high-level" trees we are interested in are, therefore, the following (from [20]):



```
GOAL Learn the session keys (allows decryption)
| 1 Decrypt RSA key exchange offline
  & 1 RSA key exchange is used
    | 1 RSA key exchange is preferred in the
        highest supported version of TLS
    | 2 Downgrade is possible to a version of TLS
        where RSA key exchange is preferred
  & 2 RSA decryption oracle (DROWN or Strong
      Bleichenbacher's oracle) is available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
```

```
GOAL Potential MITM (decryption and modification)
| 1 Force RSA key exchange by modifying ClientHello
    and decrypt it before the handshake times out
  & 1 RSA key exchange support in any TLS version
  & 2 Fast RSA decryption oracle (Special DROWN or
      Strong Bleichenbacher's oracle) available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
| 2 Learn the session keys of a long lived session
  & 1 Learn the session keys
  & 2 Client resumes the session
    | 1 Session resumption with tickets
    | 2 Session resumption with session IDs
| 3 Forge an RSA signature in the key establishment
  & 1 Fast RSA signature oracle (Strong
      Bleichenbacher's oracle) is available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
    | 4 A host with a certificate where the Subject
        Alternative Names (SAN) match this host
  & 2 The same RSA key is used for RSA key exchange
      and RSA signature in ECDHE key establishment
| 4 Private key leak due to the Heartbleed bug
```

Figure 5.4: Attack trees for leaky and tainted channels, from [20].

In the tree for tainted channels (Potential MITM) we can notice that condition 2.1 corresponds to the success of the tree for leaky channels (Learn the session keys), proving that nesting trees representing subgoals is a useful and common practice. In this notation, a pipe (‘|’) represents a logical `OR`, while an ampersand (‘&’) represents a logical `AND`.

Leaky channels

A successful tree for leaky channels means that an attacker may be able to decrypt an RSA key exchange offline, thus obtaining read access to a previous communication. As we can notice in the second branch, an RSA decryption oracle may be exploited on a host which is not the target one, but shares the same certificate or key with the target. This empirically proves both that the highly interconnected and complex nature of the modern web poses serious security threats, and that correctly managing (especially not sharing among many hosts) certificates and public keys is of uttermost importance to achieve a sufficient level of security.

Tainted channels

A successful tree for tainted channels means that an attack may set up a MITM attack, thus being able not only to decrypt and read, but also to modify the communication in real time. There are four different ways to carry out this attack, and all of them rely on the vulnerabilities at the cryptographic level presented in Section 5.1. Notice that the Heartbleed vulnerability is so disruptive that it alone my allow an attacker to run the attack.

Chapter 6

Analysis setup

In this chapter, we will explain how we carried out our analysis from a technical point of view. We will start by defining the overall structure of the tool we used (Section 6.1), presenting the tests we performed for each of the topics of interest of our analysis (Section 6.1.2). We then report the pseudocode for our complete analysis (Section 6.2) and explain the computation of attack trees in further detail (Section 6.3).
In this chapter, the reader will also understand the integration between Discovery, the tool on which the author has worked during his internship at Cryptosense, and the additional tests we implemented in order to gather more information. This integration allows us to perform thorough tests and show that many HTTPS deployments are still not satisfactory and often lack even basic security practices. The results in further detail are reported in Chapter 7.

6.1 Overall structure

Our analysis mostly relies on Discovery, a tool developed by Cryptosense for analyzing SSL/TLS and SSH deployments on given hosts. The goal is to report known vulnerabilities and misconfigurations and how to fix them.[1] The three main steps composing the analysis pipeline will be presented in the following subsections.

[1] `https://discovery.cryptosense.com`

6.1.1 Step 1: Discovering related hosts

In this phase our tool tries to discover as many hostnames related to the given one as possible. This is done by trying to resolve some default subdomains of the main host (e.g. `www.`, `idp.`, `vpn.`, ...) and checking the Certificate Transparency logs. We also use Puppeteer to catch the URLs of all the outgoing requests and check those hosts too. This is fundamental because including resources from tainted channels may compromise pages and the vulnerable hosts from which resources are included may not be found by checking subdomains and the Certificate Transparency logs (e.g. if the host is included from a generic external host).

6.1.2 Step 2: Analyzing hosts

After having discovered as many hostnames as possible, we proceed with an analysis of the discovered names in an attempt to gather all the information we need for our inferences. This is where some limitations of Discovery arose. In particular, Discovery's nature is to discover a host, perform a given set of tests on the discovered hosts and then format the results. However, we needed some more tests with respect to the standard ones computed by Discovery: for instance, Discovery does not check CSP headers nor SRI. Luckily, given Discovery's code structure, we could add the tests we needed with no significant issues. Although some features were missing, relying on Discovery allowed us to have a solid base to begin with and to avoid writing most of the logic needed, especially the one for checking vulnerabilities to attacks and computing attack trees. This greatly reduced development and testing times and allowed us to rely on an industrial-quality codebase for the most part. We will now discuss the tests we integrated and the way we integrated them, by reflecting the different categories we presented in the Introduction of this thesis.

Adoption and activation of HTTPS at the web application layer

In order to check whether HTTPS is activated, a redirect takes place and HSTS is used, we rely on Python's *requests* library and on Puppeteer. Puppeteer is used for checking whether a redirect from the HTTP to the HTTPS version of a website takes place; the *requests* library is then used to check whether the `Strict-Transport-Security` header is present, and if it is, whether `includeSubDomains` is also used.

Enforcement of the usage of HTTPS

As we have mentioned, the usage of HTTPS can also be enforced using CSP. Content-Security-Policy is a header, so we can check it while also checking for the presence of `Strict-Transport-Security` in Section 6.1.2 using the *requests* library. In particular, we parse the CSP header (if present) looking for the `upgrade-insecure-requests` and `block-all-mixed-content` directives. We also used our modified version of an open source tool[2] to check for the presence (and type) of mixed content. This is not perfect, as it does not perform thorough, recursive checks the same way a browser does; unfortunately, Puppeteer does not include Chrome's checks on mixed content as of now. The tool has proved to be quite useful (and its usage is even suggested on Google's Developers blog at [29]), although modern browsers may perform additional checks on dynamically loaded resources that this tool does not provide.

Security of cookies

The security of cookies is not taken into account in the publicly available version of Discovery: luckily, cookies too can be read from the request sent in Section 6.1.2 using the *requests* library. In order to perform the checks on the security of cookies, we rely on the conditions presented in Section 3.3.4. This step requires already having the results for the deployment of HSTS, and is therefore executed later.

[2]Available at `https://github.com/bramus/mixed-content-scan`

Safe resource inclusion

We check for the safe inclusion of resources by verifying whether SRI is deployed. This is done by gathering all the `<link>` and `<script>` tags from the page with Puppeteer and checking whether they use the `integrity` attribute. Here it is important to notice that `integrity` does *not* work for inline scripts, thus they should not be considered in this analysis. We also check, using the request from Section 6.1.2, whether CSP uses SHA in order to whitelist scripts.

Correct cryptographic implementation of the TLS protocol

This is the section in which Discovery becomes particularly useful: as a matter of fact, while working at Cryptosense the author has implemented the logic for evaluating Attack Trees, which we use to determine whether hosts might be vulnerable to attacks. All the logic needed is already present in the publicly available version of Discovery and relies on tests performed via *Nmap*[3] and *testssl.sh*[4] to compute whether hosts could be vulnerable to attacks with a decent degree of confidence. This analysis is fundamental to understand whether there are leaky or tainted channels on either the considered hosts' subdomains, main pages or related hosts. In particular, our analysis tests hosts for the following vulnerabilities, needed to compute the attack trees:

- Strong Bleichenbacher (ROBOT): needed for *tainted* and *leaky* channels. We do not consider the weak version as it is hardly exploitable.
- (Special) DROWN: needed for *tainted* and *leaky* channels
- Heartbleed: needed for *tainted* channels

Adoption of best practices in HTTPS certificates

In this analysis we are interested in understanding which kind of certificates are used: Domain Validated, Organization Validated and Extended Validated. We

[3]`https://nmap.org/`, used for DROWN, POODLE and Heartbleed
[4]`https://testssl.sh/`, used for ROBOT

implemented the logic to discern between the different types presented in Section 4.2.2, extending Discovery's logic to parse certificates.

Adoption of modern standards and headers

In order to check the presence of the `Expect-CT` header we rely on the request sent in Section 6.1.2. On the other hand, in order to check whether TLS 1.3 is supported, we used *OpenSSL*. Discovery has only recently started to support TLS1.3[5] and unfortunately we had already applied many modifications and extensions to the previous version when the new one came out, hence we implemented the TLS 1.3 check using OpenSSL.

6.1.3 Step 3: Formatting the results

Once all the tests have been run, our tool parses the result of the analysis and formats it by performing some inferences and further interpreting the obtained data. This step merges the information obtained by the many different tests run and returns a representation of each host's security condition; it also includes information on the security of its related hosts.

6.2 Complete analysis

The analysis of a single host is used as a subroutine in our code, and is used for each host in each category. The whole pipeline can be summarized as per Algorithm 1, where `discover_and_analyze_host` performs all the steps explained above.

[5]`https://cryptosense.com/blog/anssi-tls-recommendations-v1-2-in-cryptosense-discovery/`

Algorithm 1 Run tests

```
function RUN_TESTS(categories)
    $all\_results \leftarrow dictionary()$
    for all $category$ in $categories$ do
        $category\_results \leftarrow dictionary()$
        $hostnames \leftarrow hosts\_in\_category(category)$
        for all $hostname$ in $hostnames$ do
            $host\_results \leftarrow discover\_and\_analyze\_host(hostname)$
            $category\_results[hostname] \leftarrow host\_results$
        end for
        $all\_results[category] \leftarrow category\_results$
    end for
    $aggregate\_results \leftarrow compute\_aggregate\_results(all\_results)$
    return $aggregate\_results$
end function
```

6.3 Attack trees in Discovery

6.3.1 Understanding attack trees

The presence of vulnerabilities is checked by performing non-intrusive scans using open source tools such as *Nmap* and *testssl.sh*. It is fundamental to realize that a positive attack tree does not guarantee the feasibility of an attack. As a matter of fact, the computation of attack trees relies on assumptions (presented in Section 6.3.2) which may not hold. An attack may fail because of rate limiters, anomaly detection systems, etc.; still, a successful attack tree it is a strong indicator that a vulnerability is present, implying that the HTTPS deployment needs to be fixed and updated. As a matter of fact, the attack trees used in our analysis exploit vulnerabilities which have been known for years and often rely on old versions of SSL/TLS, usually SSLv2 and SSLv3. Hence, a successful tree suggests that no security updates have been applied on the host for a long time and that significant misconfiguration problems will likely be found on the host. In other words, although the host may not actually be vulnerable (e.g. thanks to firewalls or rate limiters), the success of the attack tree indicates that the host's HTTPS deployment is poorly done and should be fixed and updated anyway. Discovery also includes a "remediation" mechanism that tells the user how to fix their outdated Nginx/Apache deployment by disabling old versions of SSL.

6.3.2 Attack trees computation

Assumptions

When evaluating attack trees, some assumptions have to be made. For instance, we will never know which browser each user is using and we do not know the attacker's capabilities. When computing attack trees, realistic assumptions are being used.
The most relevant assumptions used are:

- the user's browser is modern, i.e. it does not support either SSLv2 or SSLv3;
- the user's browser does not support export ciphers and other weak algorithms such as RC4 and MD5;
- the vulnerable server can handle many parallel connections;
- the attacker can capture messages (e.g. the key exchange) in the required format to set up an attack.

Inference mechanism

The evaluation of attack trees can be seen as a simple recursive algorithm.
Starting from the root of the tree, we move deeper and deeper towards the leaves. Leaves are atomic conditions which can either be trivially evaluated, or fallback to pre-determined assumptions. Once all the leaves have been assigned a truth value (either `true` or `false`), it is possible to traverse the tree one level upwards. We thus move one level up and evaluate the intermediate nodes, whose value is computed as the result of a logical conjunction or disjunction among each intermediate node's children.

Finally, the root is computed as if it was an intermediate node and its value determines whether the attack tree is successful.

Chapter 7

Results and considerations

In this section we will present the results of our analysis. First of all, we will give some additional information regarding the scanning procedure and how we dealt with uncertainty (Section 7.1); we will then present the results for the main hosts analyzed (Section 7.2) and their related hosts (Section 7.3).
The results clearly indicate that many security techniques are not being used as widely as they should. They also prove that some websites lack even the most basic security techniques we might expect in 2020, such as a default redirect from HTTP to HTTPS, the usage of secure cookies or the lack of server-side cryptographic vulnerabilities.

7.1 Running the analysis

7.1.1 Target websites and time frame

We analyzed the top 50 websites per each of the 24 categories defined by SimilarWeb[1]. This amounts to running the tests on $24 \times 50 = 1200$ main hosts, plus all the ones discovered using our tool. The tests were run in batches using different virtual machines in order to speed up the analysis process. Analyses began on the 3rd of July 2020 and finished on the 5th of July 2020.

[1]See `https://www.similarweb.com/top-websites/`

7.1.2 Dealing with uncertainty

When running such widespread analyses many problems may arise: some hosts might be ill-configured, some others may be offline, and further ones may refuse connections or use self-signed certificates. Furthermore, the many changes we made to Discovery in order to collect all the data we needed drastically incremented the number of tests in which something could go wrong. In case nothing about a host could be analyzed, we simply did not consider the host; conversely, in case only some tests could not be performed, we adopted a reasonable fall-back strategy. For instance, in case a host performed more than 30 redirects without settling on a page nor giving useful information on its headers, we just assumed that the host did not deploy either CSP or the headers we were testing for. Analogously, in case connections were refused either on the server's end (or on our end because of a self-signed certificate) we just assumed that the headers we were looking for were not present, which in our opinion makes sense as there would be no point in deploying HSTS or the `Expect-CT` header while using a self-signed certificate.
Another important aspect of this analysis is that reproducibility is pretty hard to achieve while working on the web. Because of its ever more dynamic nature, many factors make analyses tough to reproduce: IP addresses are constantly changing because of distributed and replicated environments; HTTPS deployments may change at any given time because of choices taken from developers or site operators; hosts may go on- and off-line randomly because of network outages. We expect the general results to hold in case the analysis we carry out were to be repeated, but 100% accurate reproducibility simply cannot be achieved, unfortunately.

7.2 Main hosts

7.2.1 Secure redirect, HTTPS and HSTS

We will start presenting our results from the state of the HTTPS activation with respect to HTTP-to-HTTPS redirects and the presence of HSTS. In particular, here we evaluate whether a redirect from HTTP to HTTPS takes place, whether HTTPS is deployed with HSTS and whether HSTS is deployed with the `includeSubDomains` option. The results for the different categories can be seen in Table 7.1.

Of 1092 websites analyzed, 102 (9.3%) do not force an HTTPS redirect when trying to connect to the `http://` prefixed URL. Unfortunately, this is most browsers' default behavior, which can be avoided only by using browser extensions such as *DuckDuckGo Privacy Essentials* or *HTTPS Everywhere*. This means that the casual user may surf the website not even realizing it is not secure. Some modern browsers warn the user with an alert when he submits forms (e.g. login ones) via HTTP, but we believe serving websites over HTTPS should be the bare minimum as of 2020. This result is quite unsatisfying since avoiding plain HTTP connections has been a best practice for years.

On the bright side, 990 websites perform at least a basic HTTPS redirect. In particular, more than half of the analyzed websites (582 out of 1092, i.e. 53.3%) performs a basic HTTPS redirect, with 408 hosts (37.4%) also adopting HSTS (238 standard deployments, 170 with `includeSubDomains`). This is an interesting result, showing that site operators are slowly adopting these best practices. We were also glad to notice that the number of websites which deploy HSTS is higher than the number of websites that do not perform any form of redirect.

On a side note, none of the tested websites supports TLS1.3. Although this is understandable because TLS1.3 is fairly recent, it has been around for around a year now: browsers and web servers are mature enough to support it without major problems, hence we would expect site operators to start deploying TLS1.3 accordingly.

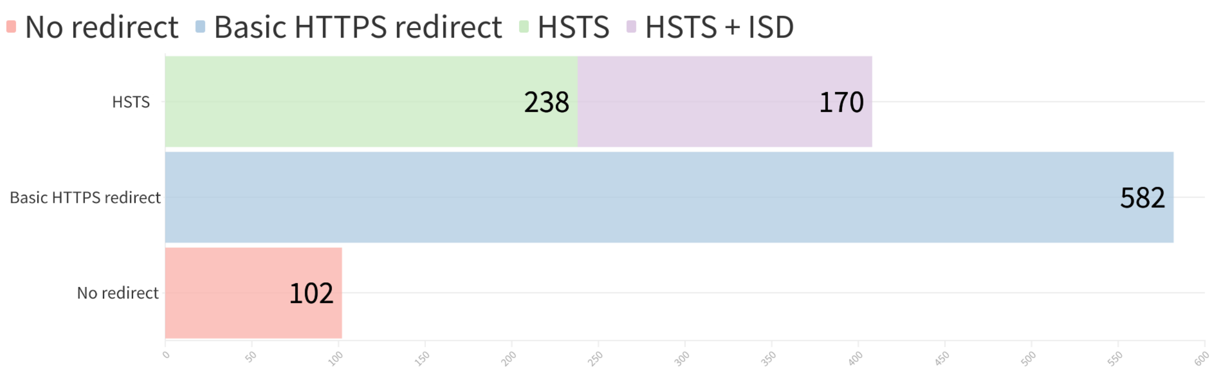


Figure 7.1: Distribution of the HTTPS activation.

7.2.2 Mixed content

In this section, we analyze the presence of passive and active mixed content in the web pages analyzed. We consider the following distinction:

- *passive mixed content*: `<img>`, `<audio>`, `<video>`, `<source>` tags;
- *active mixed content*: `<embed>`, `<iframe>`, `<link>`, `<object>`, `<script>` tags.

The results for the different categories can be seen in Table 7.2.
The good news related to mixed content is that it is not as widespread as it may have been years ago. We believe this might be due to the fact that browsers have been filtering out mixed content (and possibly marking the page as insecure) for a while now. In particular, 33 out of 1092 hosts (3.0%) included some form of passive mixed content, while 67 hosts (6.1%) included active mixed content. Although this is not a strong indicator of security problems per-se, it is generally recommended to avoid mixed content overall.

7.2.3 CSP

In this section, we analyze the usage of the `block-all-mixed-content` and `upgrade-insecure-requests` directives presented in Section 3.4.4. These directives are somewhat complementary to the deployment of HSTS, meaning that

they enforce the adoption of HTTPS and reject the usage of mixed content. The results for the different categories can be seen in Table 7.3.
Of all the main hosts analyzed, only very few use CSP. A very small number of related hosts also does, but we could find very few usages of the `upgrade-insecure-requests` and `block-all-mixed-content` directives. Also, only one of the main hosts uses CSP to perform script whitelisting. This, combined with the rare usage of SRI that will be presented in the next section, is probably one of the most significant discoveries from our analysis. There is still a long way to go in this area and we hope site operators will start deploying CSP more consistently and widely soon.

7.2.4 SRI

In this section, we report whether hosts use `script` and `link` tags with the `integrity` attribute. The `integrity` attribute, presented in Section 3.2, is extremely powerful in avoiding that an attacker tampers with included resources such as scripts. Nevertheless, it is extremely rare to find usages of this property in the wild. The results for the different categories can be seen in Table 7.4.
As per CSP, SRI is almost impossible to be found. Only 3 of the 1092 hosts analyzed include all the scripts by using the `integrity` attribute. Out of 23548 included scripts, only 75 are included using SRI. Analogously, out of 23941 `link` tags, only 26 use SRI. This result, together with the one for CSP, empirically proves that some modern techniques for additional security are extremely rare to be found in the wild. In particular, we believe that using a form of script whitelisting -be it via CSP or SRI- is fundamental to avoid the inclusion of possibly tainted files.

7.2.5 Cookies

In this section, we analyze the usage of cookies. In particular, we try to understand how many cookies are secure by means of the definition given in Section 3.3.4. We also report how many cookies are not flagged as `secure`, but still are because of an HSTS deployment. These cookies should be flagged as `secure` because in case HSTS were to be removed, they would become insecure. The results for the different categories can be seen in Table 7.5, and a visual representation is available in Figure 7.2.

We found a total of 1967 cookies distributed among 615 hosts, while the remaining 477 do not use cookies. Of the 1967 cookies, 1127 are insecure while 838 are secure. Of these 838, 71 are secure only thanks to the HSTS deployments, i.e. do not have the explicit `secure` attribute set to *true*. We believe this result is also unsatisfactory, as more than 57% of the cookies is not secure and another 3.6% would become insecure if the HSTS deployment were to be removed. Also, only 189 out of the 615 hosts using cookies adopt secure cookies only, while the remaining 426 use a mixture of secure and insecure cookies. We believe secure cookies are one of the fundamental pillars of the modern web and care when developing websites and web applications should be taken accordingly.

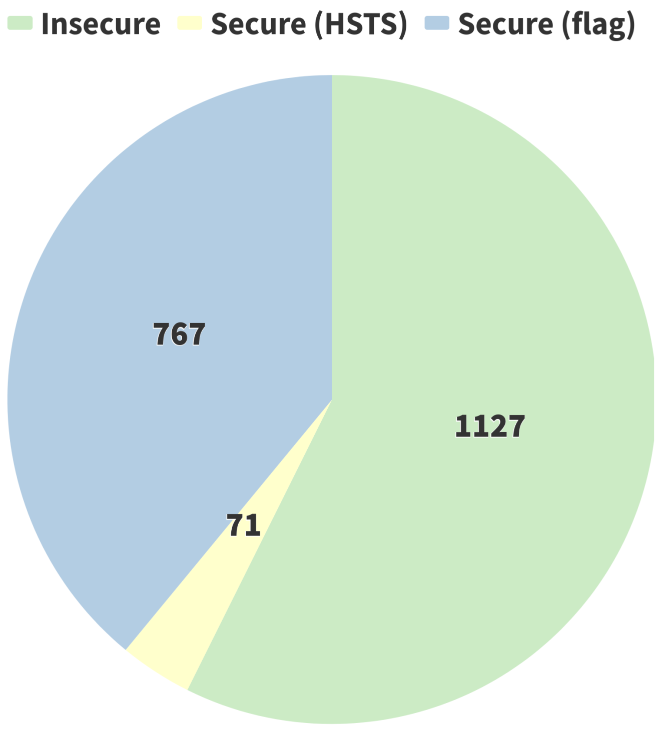


Figure 7.2: Results on the security of cookies.

7.2.6 Cryptographic implementation

In this section, we analyze the presence of vulnerabilities at the cryptographic implementation level. In particular, we look for the presence of tainted and leaky channels as presented in Section 5.1. The results for the different categories can be seen in Table 7.6.

As we mentioned in Section 5.1, the modern web tends to propagate the effect of vulnerabilities because of the interconnection of hosts and the sharing of certificates and keys. For instance, one of the hosts in the *News and media* category is vulnerable to DROWN on the 143 port (IMAP protocol), thus also making the main host insecure. In our analysis, we found 23 tainted subdomains, 29 leaky subdomains (of which 23 are leaky as a consequence of being tainted), 2 website whose homepage is tainted (and thus leaky) and 3 websites whose homepages are leaky (including the 2 tainted ones). These problems are all due to the vulnerabilities presented in Section 5.1, and can therefore be fixed by simply updating the software in the web server and its configuration, rejecting connections via deprecated protocols such as SSLv2. On the bright side, it seems like the ROBOT vulnerability is not nearly as widespread as it was just three years ago, when the authors of [63] discovered that one-third of the top 100 Alexa websites were vulnerable to the attack. Although different websites are considered, we also found that the results are better than the ones in [6] and [20] – hopefully they will keep getting better as time passes.

7.2.7 Certificates

In this section, we analyze the distribution of certificates, counting how many are Domain Validated, Organization Validated and Extended Validated. The results for the different categories can be seen in Table 7.7 and Figures 7.3 and 7.4. We also report how many hosts use the `Expect-CT` header.

Our analysis shows that 90 hosts (8.2%) use Extended Validated certificates, 625 hosts (57.2%) use Organization Validated certificates and the remaining 377 hosts (34.5%) use Domain Validated certificates. The situation is quite satisfying overall, although many websites could still seek an upgrade from OV to EV or from DV to OV. Still, this result is one of the most promising of our analysis and it shows how significant the impact of Let's Encrypt and CloudFlare is in the modern web. Furthermore, 153 out of 1092 hosts (14%) use the `Expect-CT` header, which we presented in Section 4.2.3 and believe is an important feature to have considering the ever more complex management of certificates in the HTTPS ecosystem.

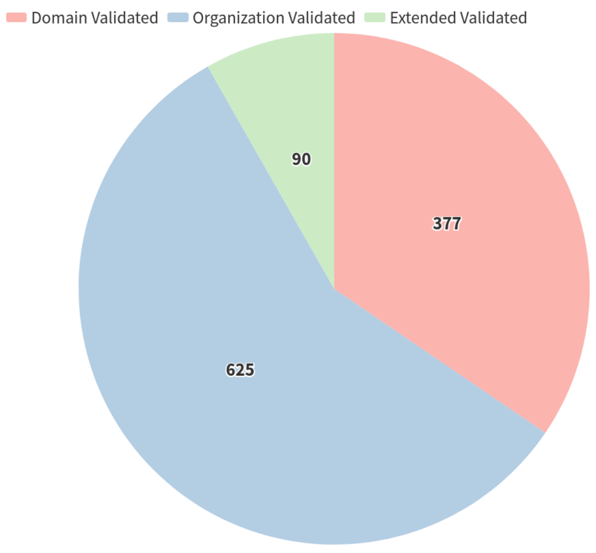


Figure 7.3: Distribution of the certificates' types.

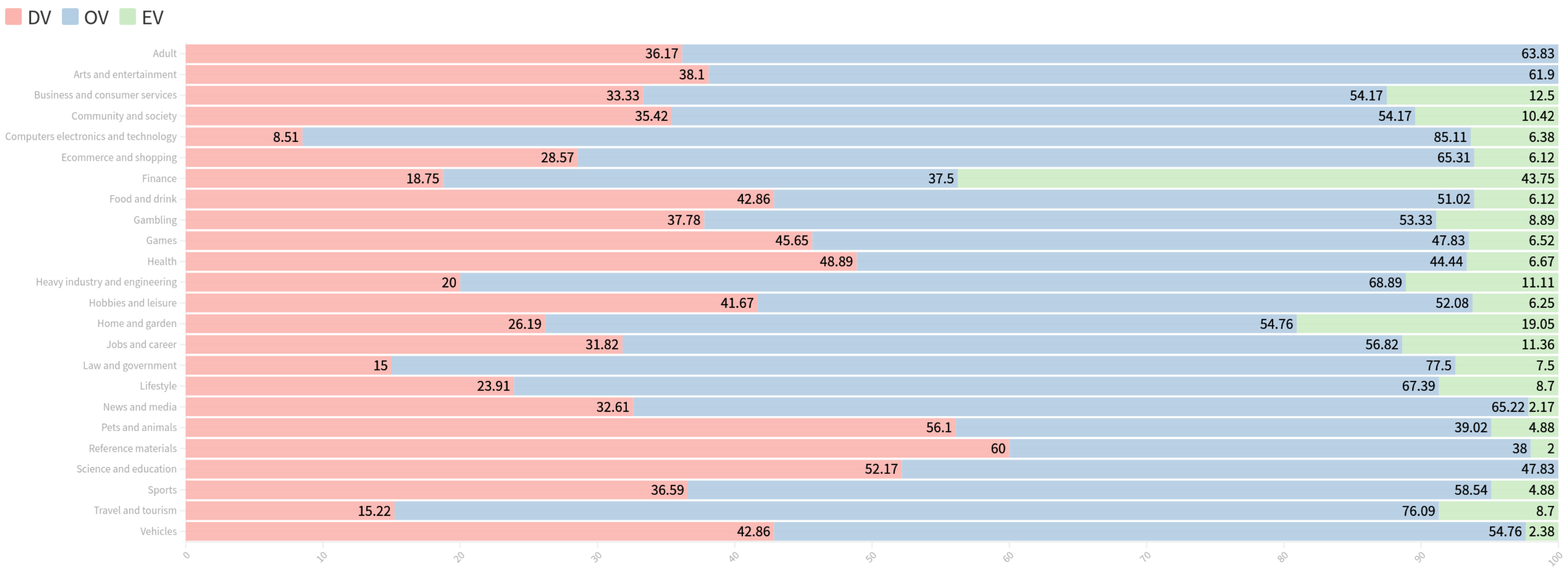


Figure 7.4: Percentages of certificates per category.

7.2.8 Summary

Overall, the state of the HTTPS deployments of the analyzed websites can not be considered fully satisfactory. In particular:

- more than 9% of the analyzed websites do not perform any redirection from HTTP to HTTPS;
- some hosts still rely on mixed content;
- the Content Security Policy headers which would be useful to enforce the usage of HTTPS and script whitelisting are not used;
- Subresource Integrity is used on too few hosts;
- more than half of the analyzed cookies are not secure;
- some hosts still support SSLv2 and are vulnerable to attacks at the cryptographic implementation level.

Although some other aspects look promising, in particular the usage of HSTS and of OV/EV certificates, we strongly believe more work could and should be put into making the analyzed hosts more secure.

7.2.9 What about categories?

One of the research questions we wanted to answer with this work was: is there a significant difference in the quality of HTTPS deployments across different categories of websites? In hindsight, finding significant differences is not an easy task; however, we can still learn some takeaways:

- there is a significant difference on the activation of HTTPS across certain different categories: for instance, 12 out of 45 hosts do not force an HTTPS redirect in the *gambling* category, yet all websites in the *vehicles* category perform at least a basic HTTPS redirect;

- only one category, *computers electronics and technology*, has a significantly higher number of secure cookies rather than insecure ones: the secure cookies here are more than double the insecure ones. Conversely, many categories (*adult*, *news and media*, *travel and tourism*, ...) have far more insecure cookies than secure ones (71-34, 52-14, 120-70);
- websites in the *finance* category are by far the ones in which we find more EV certificates (21 out of 48, i.e. 43.7%): the second highest scoring category in this ranking is *home and garden* with 8 out of 42 (19%) EV certificates.

Although noticing some particular aspects of the analysis such as these ones is interesting, they hardly carry much information regarding the overall security of the websites on their own. By comparing the results in the different tables, we can notice that there is no significant trend implying that a category is generally "more secure" than another. All in all, belonging to a specific category does not imply a higher or lower likelihood of overall security.

Category	No redirect	Redirect	Basic HSTS	include SubDomains
Adult	7/47	25/47	10/47	5/47
Arts and Entertainment	6/42	20/42	9/42	7/42
Business and Consumer Services	9/48	23/48	7/48	9/48
Community and Society	1/48	27/48	7/48	13/48
Computers Electronics and Technology	4/47	20/47	17/47	6/47
Ecommerce and Shopping	1/50	24/50	12/50	13/50
Finance	4/48	19/48	18/48	7/48
Food and Drink	2/49	30/49	6/49	11/49
Gambling	12/45	20/45	7/45	6/45
Games	7/46	21/46	11/46	7/46
Health	3/45	22/45	9/45	11/45
Heavy Industry and Engineering	4/45	27/45	12/45	2/45
Hobbies and Leisure	4/48	32/48	7/48	5/48
Home and Garden	3/42	21/42	9/42	9/42
Jobs and Career	7/44	31/44	3/44	3/44
Law and Government	7/40	14/40	14/40	5/40
Lifestyle	2/46	25/46	10/46	9/46
News and Media	1/46	29/46	11/46	5/46
Pets and Animals	4/41	28/41	6/41	3/41
Reference Materials	5/50	28/50	6/50	11/50
Science and Education	5/46	20/46	11/46	10/47
Sports	1/41	27/41	9/41	4/41
Travel and Tourism	3/46	20/46	17/46	6/46
Vehicles	0/42	29/42	10/42	3/42

Table 7.1: Results of the tests on HTTPS redirect and HSTS deployment.

Category	Passive mixed content	Active mixed content
Adult	0/47	4/47
Arts and Entertainment	2/42	2/42
Business and Consumer Services	0/48	4/48
Community and Society	0/48	2/48
Computers Electronics and Technology	2/47	3/47
Ecommerce and Shopping	0/50	0/50
Finance	1/48	5/48
Food and Drink	0/49	2/49
Gambling	1/45	4/45
Games	3/46	3/46
Health	3/45	3/45
Heavy Industry and Engineering	1/45	2/45
Hobbies and Leisure	1/48	5/48
Home and Garden	2/42	0/42
Jobs and Career	1/44	0/44
Law and Government	1/40	9/40
Lifestyle	2/46	0/46
News and Media	3/46	3/46
Pets and Animals	1/41	3/41
Reference Materials	0/50	3/50
Science and Education	0/46	5/46
Sports	7/41	4/41
Travel and Tourism	2/46	1/46
Vehicles	0/42	0/42

Table 7.2: Results of the tests on the usage of mixed content.

Category	upgrade-insecure-requests	block-all-mixed-content	script whitelisting
Adult	1/47	0/47	0/47
Arts and Entertainment	3/42	1/42	0/42
Business and Consumer Services	1/48	0/48	0/48
Community and Society	1/48	0/48	0/48
Computers Electronics and Technology	2/47	1/47	0/47
Ecommerce and Shopping	0/50	1/50	0/50
Finance	2/48	1/48	0/48
Food and Drink	0/49	0/49	0/49
Gambling	2/45	0/45	0/45
Games	2/46	0/46	0/46
Health	2/45	0/45	0/45
Heavy Industry and Engineering	3/45	0/45	0/45
Hobbies and Leisure	0/48	0/48	0/48
Home and Garden	3/42	0/42	0/42
Jobs and Career	0/44	6/44	0/44
Law and Government	2/40	0/40	0/40
Lifestyle	4/46	0/46	0/46
News and Media	3/46	5/46	0/46
Pets and Animals	1/41	0/41	0/41
Reference Materials	4/50	0/50	0/50
Science and Education	0/46	0/46	0/46
Sports	4/41	1/41	0/41
Travel and Tourism	1/46	0/46	3/46
Vehicles	1/42	0/42	0/42

Table 7.3: Results of the tests on the usage of CSP.

Category	Scripts & links not using SRI	Scripts & links using SRI
Adult	500 scripts, 1056 links	0 scripts, 0 links
Arts and Entertainment	624 scripts, 514 links	0 scripts, 0 links
Business and Consumer Services	1020 scripts, 856 links	3 scripts, 1 link
Community and Society	796 scripts, 1869 links	0 scripts, 0 links
Computers Electronics and Technology	662 scripts, 1107 links	15 scripts, 6 links
Ecommerce and Shopping	840 scripts, 973 links	0 scripts, 0 links
Finance	1118 scripts, 1072 links	3 scripts, 0 links
Food and Drink	1371 scripts, 856 links	2 scripts, 2 links
Gambling	661 scripts, 565 links	1 script, 3 links
Games	827 scripts, 749 links	3 scripts, 1 link
Health	888 scripts, 934 links	2 scripts, 0 links
Heavy Industry and Engineering	1090 scripts, 661 links	2 scripts, 1 link
Hobbies and Leisure	853 scripts, 840 links	3 scripts, 2 links
Home and Garden	1278 scripts, 734 links	5 scripts, 0 links
Jobs and Career	757 scripts, 998 links	0 scripts, 0 links
Law and Government	588 scripts, 480 links	1 script, 1 link
Lifestyle	1033 scripts, 1845 links	4 scripts, 5 links
News and Media	1388 scripts, 1181 links	4 scripts, 0 links
Pets and Animals	903 scripts, 739 links	4 scripts, 1 link
Reference Materials	984 scripts, 975 links	11 scripts, 1 link
Science and Education	832 scripts, 770 links	4 scripts, 0 links
Sports	1148 scripts, 1196 links	4 scripts, 0 links
Travel and Tourism	2266 scripts, 2244 links	0 scripts, 0 links
Vehicles	1044 scripts, 701 links	4 scripts, 2 links

Table 7.4: Results of the tests on the usage of SRI.

Category	Insecure cookies	Cookies secure thanks to HSTS	Total secure cookies
Adult	71	1	34
Arts and Entertainment	29	2	34
Business and Consumer Services	56	0	49
Community and Society	49	11	35
Computers Electronics and Technology	31	2	73
Ecommerce and Shopping	51	22	59
Finance	33	1	35
Food and Drink	68	1	31
Gambling	33	0	29
Games	35	0	25
Health	21	1	15
Heavy Industry and Engineering	28	0	26
Hobbies and Leisure	70	1	35
Home and Garden	50	2	40
Jobs and Career	84	0	28
Law and Government	18	2	17
Lifestyle	55	7	59
News and Media	52	0	14
Pets and Animals	40	1	22
Reference Materials	32	8	26
Science and Education	30	6	40
Sports	19	3	24
Travel and Tourism	120	0	70
Vehicles	52	0	20

Table 7.5: Results of the tests on the usage of cookies.

Category	Leaky main or related	Tainted main or related	Leaky subdomain	Tainted subdomain
Adult	0/47	0/47	2/47	1/47
Arts and Entertainment	0/42	0/42	1/42	1/42
Business and Consumer Services	0/48	0/48	0/48	0/48
Community and Society	0/48	0/48	1/48	0/48
Computers Electronics and Technology	0/47	0/47	1/47	1/47
Ecommerce and Shopping	0/50	0/50	0/50	0/50
Finance	0/48	0/48	2/48	1/48
Food and Drink	0/49	0/49	0/49	0/49
Gambling	0/45	0/45	1/45	1/45
Games	1/46	0/46	1/46	1/46
Health	0/45	0/45	1/45	1/45
Heavy Industry and Engineering	0/45	0/45	4/45	2/45
Hobbies and Leisure	0/48	0/48	0/48	0/48
Home and Garden	0/42	0/42	1/42	1/42
Jobs and Career	0/44	0/44	1/44	1/44
Law and Government	1/40	0/40	1/40	0/40
Lifestyle	0/46	0/46	1/46	1/46
News and Media	1/46	1/46	5/46	5/46
Pets and Animals	0/41	0/41	0/41	0/41
Reference Materials	0/50	0/50	2/50	2/50
Science and Education	0/46	0/46	0/46	0/46
Sports	0/41	0/41	0/41	0/41
Travel and Tourism	0/46	0/46	1/46	1/46
Vehicles	0/42	0/42	2/42	2/42

Table 7.6: Results of the tests on HTTPS redirect and HSTS deployment.

Category	Domain Validated	Organization Validated	Extended Validated
Adult	17/47	30/47	0/47
Arts and Entertainment	16/42	26/42	0/42
Business and Consumer Services	16/48	26/48	6/48
Community and Society	17/48	26/48	5/48
Computers Electronics and Technology	4/47	40/47	3/47
Ecommerce and Shopping	14/50	33/50	3/50
Finance	9/48	18/48	21/48
Food and Drink	21/49	25/49	3/49
Gambling	17/45	24/45	4/45
Games	21/46	22/46	3/46
Health	22/45	20/45	3/45
Heavy Industry and Engineering	9/45	31/45	5/45
Hobbies and Leisure	20/48	25/48	3/48
Home and Garden	11/42	23/42	8/42
Jobs and Career	14/44	25/44	5/44
Law and Government	6/40	31/40	3/40
Lifestyle	11/46	31/46	4/46
News and Media	15/46	30/46	1/46
Pets and Animals	23/41	16/41	2/41
Reference Materials	30/50	19/50	1/50
Science and Education	24/46	22/46	0/46
Sports	15/41	24/41	2/41
Travel and Tourism	7/46	35/46	4/46
Vehicles	18/42	23/42	1/42

Table 7.7: Results of the tests on the usage of CSP.

7.3 Related hosts and subdomains

7.3.1 A wider analysis

After having analyzed the main hosts (i.e. the 50 hosts reported in each list), we can move on to analyzing their related hosts. In this category we include their subdomains, hostnames discovered by checking the certificate transparency logs, and hosts from which the main hosts include resources such as images, stylesheets and scripts. As the reader can easily imagine, there might be tens -if not hundreds- of related hosts for each main host considered. While this is great because it leads to a much wider statistical analysis, we have to reckon that the "quality" of the discovered hosts is often lower than their "main" counterparts', thus resulting in some troubles while performing the analyses. This is a point that will need to be addressed in the future (as per Chapter 8) by making our tool more fault-tolerant. Still, the vast majority of the hosts is fine to be analyzed and we believe the results to be interesting, although the real numbers may slightly differ from the ones here presented. Unfortunately, recognizing "problematic" web pages – for instance, pages in which access is forbidden, whose content has been moved, which perform endless redirects or who never really respond to requests is not trivial at all and would require a thesis on its own. This is also the reason why we do not report as many metrics here as we did in Section 7.2.

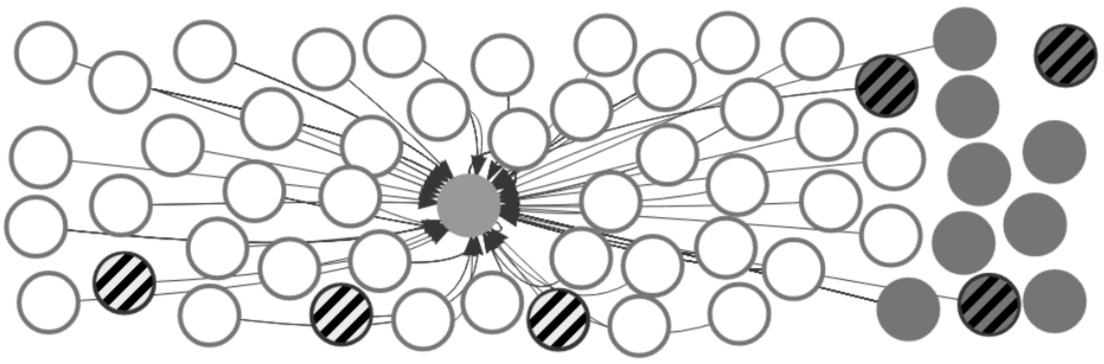


Figure 7.5: A website (central circle) can become vulnerable because of vulnerable dependencies (white striped circles) and vulnerable subdomains (gray striped circles). Taken from [20].

7.3.2 Two families

When analyzing related hosts, it is important to distinguish between HTTPS and non-HTTPS ones. As we have seen, some vulnerabilities can be found at the SSL/TLS level: this implies that they can be exploited against protocols different than HTTPS which also rely on SSL/TLS, for instance IMAPS. Also, we analyzed related hosts altogether since a significant fraction of them are shared among many different categories. In total, we discovered 62093 related hosts starting from the original 1092, of which 56933 (91.7%) are HTTPS hosts and 5160 are TLS hosts that run secure protocols different from HTTPS (IMAP, LDAP, ...).

7.3.3 Mixed content

The usage of mixed content in related hosts mostly reflects our discoveries in the main hosts case: 810 out of 56933 (1.4%) use passive mixed content and 1514 (2.7%) use active mixed content. The slightly lower percentages might be due to the fact that, empirically, many subdomains and/or related hosts consist of pages which have far less content than their respective homepages: for instance, a login page at `idp.example.com` is likely to include fewer images and scripts with respect to `www.example.com`. Further investigation in this area would be extremely interesting to be carried out.

7.3.4 CSP

The deployment of CSP, as per the "main hosts" case, is unfortunately far from common. Out of 56933 HTTPS websites we could only find 85 websites using the `block-all-mixed-content` directive, 448 hosts using the `upgrade-insecure-requests` directive and 32 websites using SHA script whitelisting. This implies that less than 1% of the considered websites use CSP – a result we cannot avoid to be unsatisfied of. Although deploying CSP is not the easiest task for site operators, we believe that some more work should be done in this direction.

7.3.5 SRI

The results for SRI too confirm that Subresource Integrity is not as widespread as it should be. Out of 341029 non-inline scripts analyzed, only 2854 had the `integrity` attribute – less than 1% of the total number. Only 199 hosts only use scripts imported with `integrity`.
Similar results are found when considering `link` tags, with 1125 links using the `integrity` attribute and 326205 links not using it on a total of 327330 tags analyzed, resulting in a quite unsatisfying 0.3% of links being protected with SRI. Although we reckon that some resources may not be really sensitive, we still believe that SRI's usage should be more common, especially considering it comes for free with some very common JavaScript frameworks: for instance, Angular.JS can build the web application with SRI by simply adding the `--subresource-integrity` flag at the build step [68].

7.3.6 Cookies

The way cookies are managed in the related hosts is fundamentally the same as that of main hosts. Out of 29371 cookies distributed among 13789 hosts, 15392 (52.4%) are not secure, with the remaining 13979 cookies being secure - 1000 of which only thanks to HSTS. A total of 4561 websites out of the 56933 analyzed ones only uses secure cookies, implying that the remaining 9228 hosts using cookies (66.9%) either uses insecure cookies, or a mixture of secure and insecure ones. These results, which can be seen in Figure 7.6, confirm the unsafe trend of not marking cookies as `secure`.

7.3.7 Cryptographic implementation and vulnerabilities

One of the main aspects we are interested in when analyzing related hosts is the presence of exploitable cryptographic vulnerabilities: as we have seen, vulnerabilities can see their effects amplified because of the numerous relationships between hosts. Here we should notice that as per the analysis carried out on main hosts, we

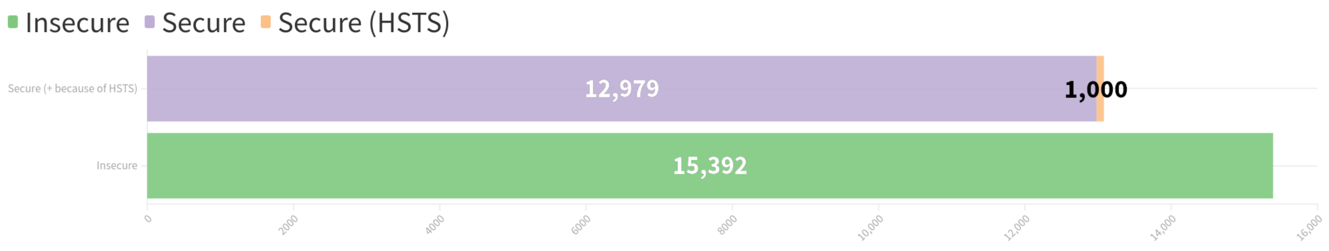


Figure 7.6: The usage of cookies in related hosts reflects their usage in the main ones.

are only considering vulnerabilities that lead to a successful attack tree; however, some non-exploitable vulnerabilities can still be found in the wild and should be fixed regardless of their "non-exploitability".

Out of 62093 hosts analyzed, we found that 3 were vulnerable to ROBOT, 109 were vulnerable to DROWN, 25 were vulnerable to POODLE and 18 were vulnerable to Heartbleed. As per Section 7.2.6, this poses serious threats on the overall security of the hosts considered. Since these vulnerabilities rely on SSLv2 and SSLv3, there should be no problem in updating the cryptographic stack, thus protecting hosts from them.

It is interesting to notice that most of the vulnerabilities are found in subdomains: 92 out of the 109 DROWN vulnerabilities, 18 out of the 25 POODLE vulnerabilities and all of the 18 Heartbleed and 3 ROBOT vulnerabilities are found on subdomains. A possible explanation may be that site operators and developers often update and improve the homepages of their hosts, but understandably do not pay the same level of attention to subdomains whose usage may be limited or only restricted to specific employees or clients.

Finally, if we collapse different vulnerable IPs and ports for the same hostnames (e.g. `https://example.com` may be vulnerable at `1.2.3.4` and `1.2.3.5`, or the same hostname may be vulnerable at different protocols), the number of vulnerable hosts becomes 95, meaning that some hosts are vulnerable at different IPs/protocols. Classic situations in which this could happen are load balancers distributing traffic between many vulnerable hosts or old setups running different services at the same vulnerable endpoint.

7.3.8 Certificates

The usage of certificates presents a situation which is not as good as the one for main hosts as far as the usage of Extended Validated certificates is concerned. This may be possibly due to the fact that site operators and developers invest more on their websites' main pages rather than less used and less visible subdomains. 1.9% of the discovered certificates is Extended Validated, compared to the 8.2% we found while analyzing the main hosts; 31.9% of the hosts use Domain Validated certificates, and the remaining 66.2% of certificates is Organization Validated.

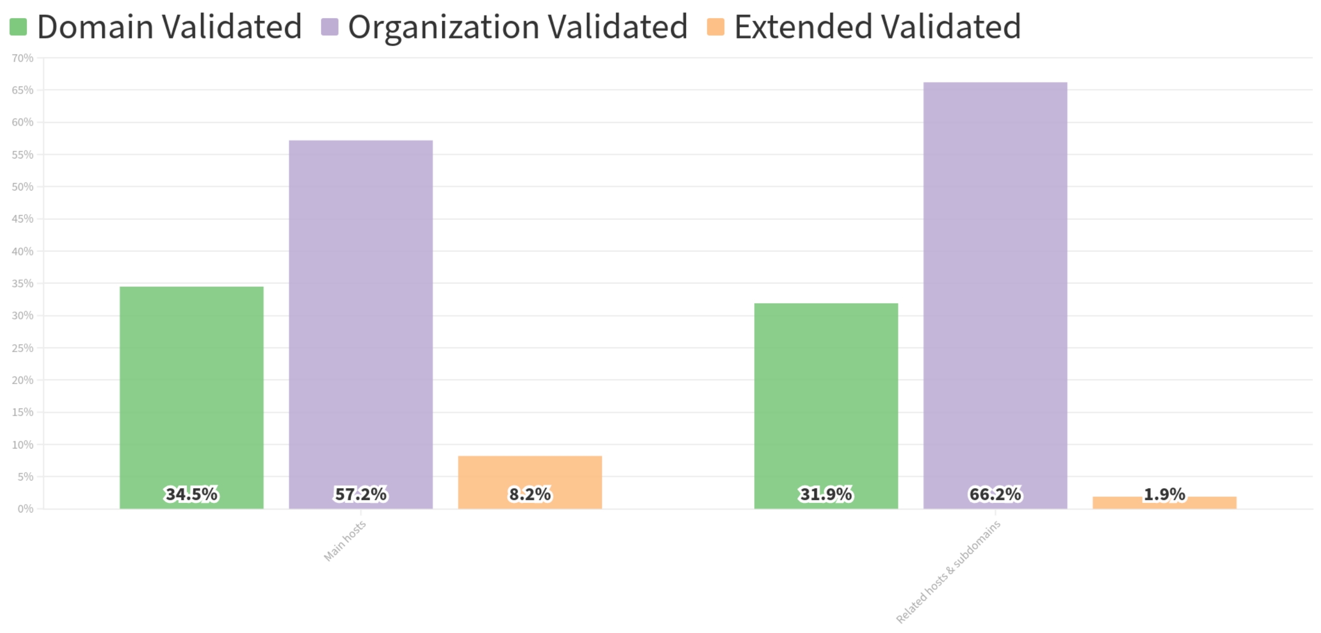


Figure 7.7: Usage of certificates in the main hosts vs. usage in their related hosts and subdomains.

7.3.9 Additional tests

We also tested related hosts for the `Expect-CT` header, discovering that 2858 out of 56933 (5%) deploy it. None of the tested websites supports TLS 1.3 yet.

Chapter 8

Future work

Many ideas for possible future development came up while working on this thesis. Here is a non-comprehensive list of them:

- *extend the analysis to further HTTPS vulnerabilities.* Discovery only tests hosts for a subset of the many existing vulnerabilities at the SSL/TLS protocol and implementation levels. It would definitely be interesting to check for the presence of more vulnerabilities.

- *create new attack trees.* This point is strongly related to the previous one: testing for more vulnerabilities would allow us to create even more attack trees and thus perform more thorough analyses.

- *make the analysis more fault tolerant.* The current toolchain is prone to errors and suffers problematic hosts. We had to exclude some hosts from our analysis because they were intractable. It is fundamental to make our tool more robust and stable in the future. Unfortunately, some of the problems we encountered were due to bugs on the severs' side or in some of the tools we used (e.g. Puppeteer). Still, we believe it is possible to improve the current levels of both stability and efficiency by adopting some smart heuristics and reviewing the way the tools interact with each other.

- *add checks related to client-side security.* Although this is a bit off-topic with respect to this thesis, it would be great to also include checks on the overall quality of the CSP deployment or on the possible presence of XSS vulnerabilities in web pages. The tool used would then become a all-in-one Swiss Army knife usable for different analyses.

- *improve checks for mixed content.* Ideally, it would be best to be able to use the mixed content detector used in modern browsers such as Google Chrome and Mozilla Firefox. In particular, we hope the functionality will be added to Puppeteer. If this is not an option, it would still be useful to improve the checks on mixed content by recursively navigating inline code and included CSS/JS files.

In the last few years, the fundamental importance of security has become tangible and highly renowned by means of many breaches and outages due to poor configuration. We hope to find more and more ideas to improve our analyses in future literature.

Chapter 9

Conclusions and final remarks

In this work, we have performed an analysis of the security the of HTTPS deployments of many different hosts. This analysis has taken into account many different factors that can (and should) be used to achieve satisfying levels of security while deploying HTTPS websites. When analyzed on their own, these factors provide interesting insights on specific possible problems of the deployments; when analyzed in combination, they provide a sufficiently accurate general picture of those same deployments' quality.

With our analysis we have discovered that, on average, websites are less prone to vulnerabilities than they were years ago, but we also found some critical problems that should have been fixed long ago and we noticed a certain reluctance towards more modern security techniques that would provide high degrees of security if used.

Correctly configuring HTTPS is, unfortunately, not easy. Many different techniques should be used and not all the available security mechanisms can always be deployed. Furthermore, as more and more security mechanisms and countermeasures become available, it is challenging to assess which ones to prioritize. However, we believe that problems such as being vulnerable to attacks which have been known for years is a problem on its own, as it indicates a lack of updates and willingness to keep a well-deployed cryptographic stack.

In conclusion, we hope that our analysis and the issues we have highlighted will encourage site operators to improve their current HTTPS deployments. While this work is not to be considered a 100% complete analysis, the trends it presents are strong indicators that there is still much room for improvement as far as security is concerned. We hope to be able to give even more significant insights on this topic in the future.

Chapter 10

Acknowledgments

First of all, I would like to thank my family and friends for another two years of support and love. There is much more than work and study in life and I am lucky to experience all of that with people I genuinely care about.
I also owe many thanks to Stefano and Alvise, whose kindness and helpfulness has made my job phenomenally pleasant and fun, and to prof. Focardi, for having given me the chance to work on this project in the first place.
Finally, Bertrand, Graham and all my ex-colleagues at Cryptosense deserve a special mention for having patiently guided me throughout my internship, making me a better software engineer and person overall.

Bibliography

[1] Cloudflare. What Is SSL? — SSL and TLS. https://www.cloudflare.com/learning/ssl/what-is-ssl/, Last visited: 28th April 2020.

[2] T. Polk S. Turner. RFC6176. prohibiting secure sockets layer (ssl) version 2.0. https://tools.ietf.org/html/rfc6176, Last visited: 28th April 2020.

[3] A. Pironti A. Langley R. Barnes, M. Thomson. RFC7568. deprecating secure sockets layer version 3.0. https://tools.ietf.org/html/rfc7568, Last visited: 28th April 2020.

[4] Chrome Platform Status. Tls 1.0 and tls 1.1 (removed). https://chromestatus.com/feature/5759116003770368, Last visited: 28th April 2020.

[5] ANSSI (Agence nationale de la sécurité des systèmes d'information). Recommandations de sécurité relatives à tls. https://www.ssi.gouv.fr/particulier/guide/recommandations-de-securite-relatives-a-tls/, Last visited: 8th June 2020.

[6] Stefano Calzavara, Riccardo Focardi, Alvise Rabitti, and Lorenzo Soligo. A hard lesson: Assessing the https deployment of italian university websites.

[7] Mozilla MDN Web Docs. Confidentiality, Integrity, and Availability. https://developer.mozilla.org/en-US/docs/Archive/Security/Confidentiality,_Integrity,_and_Availability, Last visited: 28th April 2020.

[8] SecGroup Ca' Foscari. Introduction to cryptography. https://secgroup.dais.unive.it/teaching/cryptography/introduction/, Last visited: 8th June 2020.

[9] Mozilla MDN Web Docs. HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP, Last visited: 28th April 2020.

[10] Mozilla MDN Web Docs. Content-security-policy. https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages, Last visited: 29th April 2020.

[11] Mozilla MDN Web Docs. An overview of http. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview, Last visited: 30th April 2020.

[12] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1323–1338, 2017.

[13] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS Using SSLv2. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, pages 689–706. USENIX Association, 2016.

[14] Hanno Böck, Juraj Somorovsky, and Craig Young. Return Of Bleichenbacher's Oracle Threat (ROBOT). Cryptology ePrint Archive, Report 2017/1189, 2017. Online, cit. [2018-10-29].

[15] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar R. Weippl. "i have no idea what i'm doing" - on the usability of deploying HTTPS. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1339–1356, 2017.

[16] Salvatore Manfredi, Silvio Ranise, and Giada Sciarretta. Lost in tls? no more! assisted deployment of secure TLS configurations. In *Data and Applications Security and Privacy XXXIII - 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15-17, 2019, Proceedings*, pages 201–220, 2019.

[17] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback, 2014. Online, cit. [2018-10-29].

[18] Boris Rogier. 3 things you should know about https, ssl/tls traffic with wireshark. https://www.networkdatapedia.com/post/3-things-you-should-know-about-https-ssltls-traffic-with-wireshark, Last visited: 28th April 2020.

[19] Hugo Krawczyk, Kenneth G Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In *Annual Cryptology Conference*, pages 429–448. Springer, 2013.

[20] Stefano Calzavara, Riccardo Focardi, Matús Nemec, Alvise Rabitti, and Marco Squarcina. Postcards from the post-HTTP world: Amplification of HTTPS vulnerabilities in the web

ecosystem. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 281–298, 2019.

[21] Wazen Shbair, Thibault Cholez, Jérôme François, and Isabelle Chrisment. A multi-level framework to identify https services. 04 2016.

[22] Microsoft Docs. Cipher suites in TLS/SSL (Schannel SSP). `https://docs.microsoft.com/en-us/windows/win32/secauthn/cipher-suites-in-schannel`, Last visited: 26th April 2020.

[23] Mozilla Wiki. Security/server side tls. `https://wiki.mozilla.org/Security/Server_Side_TLS`, Last visited: 30th April 2020.

[24] The White House. Administration of export controls on encryption products. `https://www.govinfo.gov/content/pkg/FR-1996-11-19/pdf/96-29692.pdf`, Last visited: 29th April 2020.

[25] Cloudflare. Why use tls 1.3? — ssl and tls vulnerabilities. `https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/`, Last visited: 2nd May 2020.

[26] E. Rescorla. The transport layer security (tls) protocol version 1.3. `https://tools.ietf.org/html/rfc8446`, Last visited: 2nd May 2020.

[27] Robert van Rijn. Whats new with tls 1.3. `https://medium.com/@vanrijn/what-is-new-with-tls-1-3-e991df2caaac`, Last visited: 2nd May 2020.

[28] Mozilla MDN Web Docs. Strict-transport-security. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security`, Last visited: 28th April 2020.

[29] Jo el van Bergen. What is mixed content? `https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content`, Last visited: 29th April 2020.

[30] Carlos Joan Rafael Ibarra Lopez (Chrome security team) Emily Stark. No more mixed messages about https. `https://blog.chromium.org/2019/10/no-more-mixed-messages-about-https.html`, Last visited: 29th April 2020.

[31] Mozilla MDN Web Docs. Same-origin policy. `https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy`, Last visited: 27th April 2020.

[32] Stefano Calzavara. Slides on *Same Origin Policy*. `https://secgroup.dais.unive.it/wp-content/uploads/2020/02/sop.pdf`, Last visited: 27th April 2020.

[33] Mozilla MDN Web Docs. Cross-Origin Resource Sharing (CORS). https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS, Last visited: 28th April 2020.

[34] Mozilla MDN Web Docs. Subresource integrity. https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity, Last visited: 28th April 2020.

[35] Mozilla MDN Web Docs. HTTP cookies. https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies, Last visited: 26th April 2020.

[36] Stefano Calzavara. Slides on *Session Management*. https://secgroup.dais.unive.it/wp-content/uploads/2020/02/sessions.pdf, Last visited: 27th April 2020.

[37] A. Barth. HTTP state management mechanism. https://tools.ietf.org/html/rfc6265, Last visited: 27th April 2020.

[38] David Johansson. Cookie security. myths and misconceptions. video with demo. https://https://www.youtube.com/watch?v=ND_IaksBRQE, Last visited: 27th April 2020.

[39] Rowan Merewood. SameSite cookies explained. https://web.dev/samesite-cookies-explained/, Last visited: 27th April 2020.

[40] M. West. Same-Site cookies, rfc6265bis. https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site-00, Last visited: 27th April 2020.

[41] Stefano Calzavara, Alvise Rabitti, Alessio Ragazzo, and Michele Bugliesi. Testing for integrity flaws in web sessions. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, pages 606–624, 2019.

[42] David Johansson. Cookie security. myths and misconceptions. https://owasp.org/www-chapter-london/assets/slides/OWASPLondon20171130_Cookie_Security_Myths_Misconceptions_David_Johansson.pdf, Last visited: 27th April 2020.

[43] Mozilla MDN Web Docs. Content-security-policy. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy, Last visited: 29th April 2020.

[44] Positive Technologies. Web applications vulnerabilities and threats: statistics for 2019. https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/, Last visited: 29th April 2020.

[45] Sean Michael Kerner. Hackerone reports bug bounties rise as xss remains the top flaw. `https://www.eweek.com/security/hackerone-reports-bug-bounties-rise-as-xss-remains-the-top-flaw`, Last visited: 29th April 2020.

[46] Stefano Calzavara. Slides on *XSS*. `https://secgroup.dais.unive.it/wp-content/uploads/2020/02/xss.pdf`, Last visited: 29th April 2020.

[47] Cloudflare. What is cross-site scripting? `https://www.cloudflare.com/learning/security/threats/cross-site-scripting/`, Last visited: 29th April 2020.

[48] Stefano Calzavara. Slides on *Content Security Policy*. `https://secgroup.dais.unive.it/wp-content/uploads/2020/03/csp.pdf`, Last visited: 30th April 2020.

[49] Stefano Calzavara. Slides on *HTTPS*. `https://secgroup.dais.unive.it/wp-content/uploads/2020/03/https.pdf`, Last visited: 30th April 2020.

[50] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. Semantics-based analysis of content security policy deployment. *TWEB*, 12(2):10:1–10:36, 2018.

[51] Thales. What is public key infrastructure (pki)? `https://cpl.thalesgroup.com/faq/public-key-infrastructure-pki/what-public-key-infrastructure-pki`, Last visited: 1st July 2020.

[52] SSH.com. Pki - public key infrastructure. `https://www.ssh.com/pki/`, Last visited: 1st July 2020.

[53] ScienceDirect. Certificate validation. `https://www.sciencedirect.com/topics/computer-science/certificate-validation`, Last visited: 20th June 2020.

[54] Digicert Inc. Certificate profiles. `https://www.digicert.com/wp-content/uploads/2018/01/Certificate-Profiles.pdf`, Last visited: 29th April 2020.

[55] Mozilla MDN Web Docs. Expect-ct. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expect-CT`, Last visited: 30th April 2020.

[56] Certificate Transparency. How certificate transparency works. `https://www.certificate-transparency.org/how-ct-works`, Last visited: 22nd June 2020.

[57] CheapSSL. What is certificate transparency (ct)? how does it work? `https://cheapsslsecurity.com/blog/what-is-certificate-transparency-ct-how-does-it-work/`, Last visited: 22nd June 2020.

[58] Cisco. What are the most common cyber attacks? https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html, Last visited: 29th April 2020.

[59] Tomasz Andrzej Nidecki. What is the poodle attack? https://www.acunetix.com/blog/web-security-zone/what-is-poodle-attack/, Last visited: 8th July 2020.

[60] B. Moeller and A. Langley. Rfc 7507: Tls fallback signaling cipher suite value (scsv) for preventing protocol downgrade attacks. https://tools.ietf.org/html/rfc7507, Last visited: 29th June 2020.

[61] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 1–12, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[62] NIST. Cve-2016-2107 detail. https://nvd.nist.gov/vuln/detail/CVE-2016-2107, Last visited: 29th April 2020.

[63] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of bleichenbacher's oracle threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*, pages 817–849, Baltimore, MD, August 2018. USENIX Association.

[64] E. Rescorla T. Dierks. The transport layer security (tls) protocol version 1.2. https://tools.ietf.org/html/rfc5246, Last visited: 29th April 2020.

[65] Inc. Synopsys. Heartbleed. https://heartbleed.com/, Last visited: 2nd May 2020.

[66] M. Williams R. Seggelmann, M. Tuexen. Transport layer security (tls) and datagram transport layer security (dtls) heartbeat extension. https://tools.ietf.org/html/rfc6520, Last visited: 2nd May 2020.

[67] FenixFeather. Simplified heartbleed explanation. https://commons.wikimedia.org/wiki/File:Simplified_Heartbleed_explanation.svg, Last visited: 2nd May 2020.

[68] Fabrizio Fortunato. Angular subresource integrity. https://izifortune.com/angular-subresource-integrity/, Last visited: 8th July 2020.