Internet Security Scanner

An extendable internet security scanner and analyser

Kristian Helgesen Torkveen

Thesis submitted for the degree of
Master in Programming and System Architecture:
Information Security
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

Internet Security Scanner

An extendable internet security scanner and analyser

Kristian Helgesen Torkveen

Internet Security Scanner

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The use of computers and networks in our daily lives is a fact at this point, and it is difficult to avoid interacting with computer systems and the internet. As more of our life depend on transmitting data over the internet, it's important that we can facilitate this data transmission in a secure manner. In short, this means ensuring that developers and applications have access to use secure encrypted protocols for communication, and that these protocols are kept up to date and used in a proper manner.

For the World Wide Web (WWW) and Hypertext Transfer Protocol (HTTP) the solution to facilitate this secure data transmission has become the Transport Layer Security (TLS) encrypted protocol, but the unencrypted HTTP protocol is still in use. Additionally, HTTP clients and servers implement several protocol specific security measures in the form of HTTP headers.

This presents a challenge. Considering the decentralized nature of the world wide web and the various independent servers hosting web applications, how can we monitor the adoption and support of various versions of the TLS protocol and support for security features? In this thesis, I have created a scanner to monitor the use of TLS versions and various security features in the TLS and HTTP protocols. The scanner is be designed to be extendable in order to allow collection of more data and analysis of collected data.

Contents

List of Figures

List of Tables

Acknowledgements

I want to thank my thesis supervisor at UiO Nils Gruschka for his support and guidance while writing this thesis.

I also want to thank my family, friends, and those close to me for their continued support during my studies.

Chapter 1

Introduction

1.1 Motivation

We use more and more computers and networks in our daily life, and avoiding these interactions using the internet is difficult. If someone don't personally access the internet or a computer, some organisation they interact with will do so. Personal communications, private businesses and public government services are all online or moving there, and most forms on paper will at some point be digitized. If we pay for a product with our card, call a friend on the phone, or catch up on news the information needed will likely be sent through the internet, where it can pass through various unknown networks with unknown owners and equipment. Even the content of a physical paper newspaper is likely to have been sent over the internet at some point during it's creation.

Hypertext Transfer Protocol (HTTP) has become the standard for a lot of our internet use, primarily relying on Transport Layer Security (TLS) to prevent man-in-the-middle attacks (MITM). TLS is a protocol which provides the necessary confidentiality and integrity through encryption. The confidentiality aspect means ensuring that the network and equipment the message passes through is unable to read the content of the message, while the integrity aspect means ensuring that any attempted modification of the message can be discovered by the recipient. The recipient can then discard the modified message. These qualities are fundamental to many aspects of online life.

These are not just theoretical threats, real world attacks that have been done at scale. Hotel wifi and ISPs have injected advertisements into websites [55][28], airlines inject flight information into websites [33][24], and even nation states have used this kind of man-in-the-middle attack [32]. It can also be used in combination with other attacks, in order to route more traffic through the networks doing the man-in-the-middle attacks [15]. While not every attack mentioned here is nefarious, they are all still worrying as they use methods which can just as well be used to monitor the user without their consent, read sensitive information from websites, or even trick the user.

All specifications and software can have flaws, leading to bugs and

vulnerabilities which can enable attacks man-in-the-middle and other attacks, so TLS and implementations of TLS need updates to be secure. All of these factors makes it important for us to know the current state of TLS. Which versions are used by websites? How many websites use various security features built into TLS and HTTP?

1.2 Contribution

While scanners analysing TLS use and settings exist, they often offer either limited options for anyone to do further analysis of the raw data or no way for outside contributors to extend the scan to collect more data. When only summaries of data is available this is of limited utility, as you often cannot make connections between various datapoints. This can make it hard to answer questions that examine these connections, for example when examining the use of security features for each TLS version, or which top level domains tend to have stronger settings.

In this thesis, I will create an extendable scanner which can collect data regarding it's connections to websites using TLS at scale to help answer questions around the use of HTTP over TLS (HTTPS). This scanner will collect information about which TLS versions are supported by a website and whether a series of TLS and HTTP security features are utilized by the website, then save the collected data in a database. This database can then be used to generate a scan report or for further analysis.

Enabling further analysis is also a core part of the scanner project. The project will also contain a web application with a json based REST api which enables easy export of data as well as easy execution of queries in the database. The web application will also contain a simple report of each completed scan, which can be used to get an overview of the results. This gives easy access to the data from many programming languages, as many languages have access to a HTTP client and a json library.

This is important because specifications and software are not perfect [26], and cannot cater to every use case without configuration. We do sometimes need to update implementations of TLS or even the TLS protocol itself to fix potential flaws, or add additional security features. These features might additionally allow or require further configuration. This opens us up to websites having misconfigured software or running old versions. Collecting more information about this allows us to measure adoption of new versions and features better, which in turn can be used to improve the design and implementation of future updates.

Chapter 2

Background

This chapter describes some of the technology used on the internet today relevant to this topic, as well as previous research, and some known vulnerabilities.

Hypertext Transfer Protocol (HTTP) is a stateless application level network protocol for sending information to and loading information from remote servers. It allows a client to send a request and receive a response, depending on the content of their request. A request contains a verb, such as GET or POST, along with other header fields, and potentially a body with information in it. Most header fields in use have been given special meanings, but an application developer can also create their own. A response contains a status, headers, and can also contain another body of data for the client [46]. A request and its response can be used to fetch information, update information, or trigger some kind of action in the target application. Examples of this is to register an account on a website (update), fetch your account information, and to request a password reset email (trigger action).

Transport Layer Security (TLS) is a cryptographically secure, application independent, reliable network communication protocol. It's primary goals are to provide privacy and data integrity between two applications. This means that any application developer should be able to use TLS in their applications, and to build secure protocols on top of TLS. Additionally, TLS itself is extendable, and anyone can build application protocols and extensions on top of TLS [47].

TLS is fundamental to the security of many systems today. 88% of page loads by Google Chrome Windows users utilized HTTPS (HTTP over TLS), as of the 2nd of May 2020 [23]. For a growing amount of services, including healthcare and banking services, HTTPS is the primary way to use them. This can, for example, be through web interfaces or through application programming interfaces, and these services then rely on TLS to keep the information sent between the user and the service secure. This makes the security of TLS very relevant, so that we can ensure that the services our society relies on remain secure.

All software has bugs and vulnerabilities. Since 2014, Google Project Zero has found 1808 bugs listed on their public bug tracker [26], including

several vulnerabilities affecting implementations of TLS. The importance of TLS for the security of our systems also means we need to be aware of these vulnerabilities, as well as to what extent these vulnerabilities are patched in systems. To get an overview of this, it's useful to scan and aggregate the status of a large amount of websites, using lists such as Alexa Top Sites [3] or Tranco [30]. Knowing what vulnerabilities are common and to what extent patches have been installed is a useful step towards improving the patching process and knowing whether all available patches work as intended.

2.1 Certificates

TLS uses X.509 certificates and public key infrastructure for its authentication handshake protocol [47].

Public Key Infrastructure refers to a hierarchical tree of signed certificates. A trusted entity can create a root certificate, and use this to sign other certificates. These certificates may then be able to sign other certificates again (an intermediate certificate), or they may only be used to identify a host (a leaf certificate). A set of trusted certificates can then be stored on the relevant hosts [11]. To validate a certificate, we can then see which certificate was used to sign it, and follow the path until we reach a trusted certificate or the path runs out. If we find a trusted certificate, we can validate the identity, otherwise we should not trust it.

There are multiple certificate types used for HTTPS, providing different kinds of identity assurance [11] [7]. While there are more, I will primarily look at Domain Validation certificates (DV) and Extended Validation certificates (EV) certificates in this thesis. A domain validation certificate should validate that a host is permitted to serve a response to certain host names, whether that hostname is a domain name or an IP address [11]. For example, this allows the server serving example.com to prove that it is the legitimate example.com, rather than some other device attempting to trick the client application. An Extended Validation certificate takes this a step further, by also validating that a domain name owner actually legally exists, and that the domain name belongs to them [7] [16]. It is, however, important to note that this does not prove that the owner does legitimate business, nor are business names unique accross different countries [36].

2.2 An overview of a TLS connection

At the heart of TLS is the TLS Record Protocol, which has several protocols layered on top of it, including the TLS Handshake Protocol and the TLS Application Data Protocol. The record protocol's task is to take the messages to be transmitted and fragment them, compress them, sign them, and encrypt them before transmitting the result. On the receiving end, it will decrypt the data, verify signatures, decompress, and reassemble the data before delivering it to the higher level client. The fragmentation, compression, signatures, and encryption used is defined by the current

shared connection state. The handshake and application data protocols are built on top of the record protocol, and are both higher level clients of the record protocol. [47].

The TLS Handshake Protocol is performed when a client and a server first start communicating, setting the shared connection state and optionally authenticating each other. First they exchange hello messages agreeing on algorithms, and exchange random values. They will then exchange the necessary cryptographic parameters to agree on a premaster secret value. At this point they may also exchange certificates, allowing the client and the server authenticate themselves. To finish the handshake they provide the security parameters to their record protocols, and verify that their peer has the same security parameters and that the handshake was not tampered with by an attacker [47]. After this is complete, they should have a secure connection to each other.

2.3 Related work

There are some services doing similar work today. The Qualys SSL Labs SSL Pulse monitors and provides snapshots of the TLS support and security for the top 150 000 websites on the Alexa Top Sites list [44]. Independent researcher Scott Helme does daily crawls of the top 1 000 000 websites on the Tranco list [20], checking various security metrics and providing the raw data, daily statistics, and a more in depth report every 6 months at crawler.ninja [17].

Qualys SSL Labs SSL Pulse provides a good overview, but lacks the option to do further analysis of the data or check other mechanisms than those included. You are also unable to get more frequent data than their monthly scans. All of this limits the usefulness of the service for many research purposes.

Scott Helme, on the other hand, does provide raw data downloads which can be used for further analysis, if his crawler checks for what you are interested in. The closed source nature of the crawler limits it's extensibility, other developers cannot easily add to it.

This leaves a void which I intend to fill. By creating an extensible scanner, with both raw data and the source code available, I can create a valuable research tools for other researchers. On it's own, it can provide a useful overview, and provide in depth data about single websites for further analysis. When combined with existing solutions, such as Scott Helme's crawler.ninja, we can get raw data and start to draw connections between the different datasets.

2.4 Common attacks

There are some attacks which are common against TLS, and several of the vulnerabilities I will mention utilize these as a part of a larger attack. While these patterns are of limited use against TLS on their own, they can be used to enable other attacks.

A man-in-the-middle (MITM) attack is when an attacker can access or control the connections between two peers, for example a client and a server. The attacker can then see and modify the data sent between the client and the server. With a secure TLS connection, this is not a problem, as certificate signature validation, handshake anti-tamper mechanisms, and data encryption can prevent insight into the actual data as long as failed certificate validation is not ignored. However, it still gives an attacker a way to manipulate what is sent, or even prevent it from arriving [8]. If a part of the connection is unencrypted, a MITM attack can be used to change the information sent, which can sometimes prevent a change to an encrypted channel completely. For example, when a website redirects a user from HTTP to HTTPS, an attacker could prevent that redirect and serve the website unencrypted, giving the attacker full access to what the client sends and receives.

A version downgrade attack can make exploitation of older vulnerabilities possible, by somehow causing the connection to use an older version of TLS or even dropping encryption completely. An common way to do this is to block the connection using a man-in-the-middle attack, and hope that the client will fall back to less secure settings and attempt to connect again, which many devices do in order to support legacy servers [2] [35]. Alternatively, an attacker can attempt to use application specific ways to influence the security settings used. Making a target use older versions or weaker settings can be very valuable for an attacker, as it can allow the exploitation of known vulnerabilities in these older versions to spy on the content of the encrypted connection, either by seeing the content itself or by being able to access more information about it than they could otherwise.

An oracle is a mechanism that will expose some information that is not ordinarily available, and some some of the mentioned attacks will utilize these. In cryptography, this can often be some unintended side effect caused by the implementation, exposing information. For example, when attempting to send arbitrary data to a server that would attempt to decrypt the received data, it might respond slightly differently depending on the type of error caused when it attempts to decrypt the arbitrary data, which can expose information about whether some property of the arbitrary data might be correct. If an attacker knows which properties might be correct and the data they sent, they might then be able to use the information the oracle exposes to break the algorithm or as a part of a larger attack.

2.5 Checking TLS security

In this section I will write about some of the security mechanisms and vulnerabilities we will be checking for.

2.5.1 Security mechanisms

Cryptography

The TLS standard specifies several cipher suites [47], which are sets of cryptographic algorithms used for the key exchange, bulk encryption, and verifying the integrity of a received message. The key exchange algorithm will typically be using a asymmetic public key cryptographic algorithm, meaning different keys are used to encrypt and decrypt the data. After these keys have been established, the peers will exchange a symmetric bulk encryption key, which is more efficient at encrypting large amounts of data. Finally, a hashing algorithm is used along with cryptographic signatures to verify the integrity of a received message [50].

Selecting secure algorithms and using sufficiently long keys is of course important, in order to ensure that communication over TLS is secure. I will not argue about which algorithms and key lengths are secure in this thesis, but I will refer to existing standards on the matter, such as the American NIST SP 800-57 revision 5 [4][1].

HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) is a method to make web browsers default to HTTPS when connecting to a server, and refuse to connect over plain HTTP [27]. Ordinarily, a web browser will try to connect using HTTP by default, and then often be redirected to HTTPS. This initial request can be hijacked by an attacker in a MITM attack (see 2.4), so being able to default to HTTPS is preferable. Strictly speaking, this is a security feature in HTTP rather than TLS, but it is a relevant case of an application adding an additional security measure to prevent attacks against the initial plain HTTP request.

HSTS works by setting a HTTP header, which causes the browser to save that it may not use plain HTTP to reach the website for a given amount of time [27]. This means that after the first connection a browser can skip the insecure first HTTP request, which often only redirects to HTTPS regardless. In addition, a website administrator can have their website preloaded [22] [25], meaning that it will be added to the list when the user installs a web browser. This eliminates the initial insecure HTTP request for the first connection to a website as well.

OCSP stapling

There are several ways to check if a certificate has been revoked. The most well known ones are Certificate Revocation Lists (CRL) and Online Certificate Status Protocol (OCSP). A CRL is a signed list of serial numbers of certificates which have been revoked, which the client can download and then look up certificates in. The problem with these lists is that they get large and difficult to manage and distribute to the clients. Over

[1] Can be downloaded from https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final

time, they grow to require a significant amount of bandwidth to distribute to all the devices connected to the internet. As a solution, OCSP was introduced, letting the client check the status of a single certificate by contacting an OCSP responder [54] [18]. The OCSP response from the certificate authority contains timestamps describing it's validity and is cryptographically signed to prove its legitimacy [49]. This also had some problems, however, and OCSP stapling was presented as a new solution.

The original way to deliver the OCSP response is generally unreliable [19] [42]. It is that the client would contact the OCSP responder directly for it. However, this means that if the OCSP responder is down the client would not get any response, and if it is slow, it might delay the client's connection by as much as several seconds. As a result, clients will typically ignore it if there is no response, or it takes too long. This also opens up for an attacker performing a MITM attack to block OCSP, so that the client will not be aware that a certificate has been revoked [48] [39].

OCSP stapling provides way for the client to request that the server adds the OCSP response when sending it's certificate, as opposed to the client contacting the OCSP responder itself [39]. This means that the server can cache it, so that a client connection requires less roundtrips to other servers and it causes less load on the OCSP responder. It also increases the reliability of the OCSP revocation checks, making it so that OCSP responder downtime is less likely to impact the revocation check. The fact it uses the same signed response as the client could get from the OCSP responder also ment that very little new infrastructure was necessary to implement it.

Certificate Authority Authorization

Certificate Authority Authorization (CAA) is a mechanism that allows site owners to specify which certificate authorities are allowed to issue certificates for their website, in a DNS record. A certificate authority is required to check this record before issuing a certificate, and only issue the certificate if they are listed [21]. While it does not completely prevent a malicious certificate being issued, it can prevent certificates being issued without proven control of the website due to bugs in the interface of other certificate authorities, and similar issues [9].

Certificate Transparency

Certificate Transparency (CT) refers to the logging of issued certificates in public certificate logs, combined with browsers now requiring issued certificates to be logged to consider them valid [51] [29]. When a certificate is sent to a log, the log will return a Signed Certificate Timestamp (SCT), which will be stored in or with the certificate. When a client connects to a server the server will return the SCT along with the certificate, so that the client may validate the signature using the key for the relevant log [29].

Certificate Transparency allows anyone to audit the certificates issued by a certificate authority. This allows service owners to monitor for

certificates issued for their domains, so that they may notify the certificate authority about any unknown certificates and have them revoked. This makes it possible to hold certificate authorities responsible for incorrectly issued certificates [29] [10].

Protocol Downgrade Defense

Protocol downgrade defense is a way for a client to indicate that it is attempting to reconnect with weaker security than it can potentially support. The way it works is that the client adds a "false" cipher suite to it's list of supported cipher suites, if it is currently retrying the connection with weaker settings due to a previous connection failure. If the server supports protocol downgrade defense and supports a more secure version of TLS, it can then instruct the client to retry with more secure settings and end the connection, letting the client reconnect with more secure settings again. If the server does not support it, it will assume that it is an unsupported cipher suite, and ignore it [35].

2.6 The Tranco list

Tranco is a research-oriented list of the top 1 000 000 websites, freely available from tranco-list.eu [30]. It is a reproducible list which is also hardened against potential manipulation by malicious actors. The latest Tranco list is always available from tranco-list.eu/top-1m.csv.zip, and there are also links to the lists for other days on their website. It is the recommended basis for scans of the top X websites with this tool.

Chapter 3

The data collection method

In this chapter I will list and explain the data the scanner collects, and their use.

3.1 HTTP responses

Browsers still generally attempt to connect using unencrypted HTTP by default, rather than using TLS encrypted HTTPS. In these cases, the server needs to give a HTTP response directing the browser to use a TLS encrypted connection instead.

The primary way of doing this is by returning a redirect response [45]. In this regard, the scanner checks whether the server redirects the browser to load the website using HTTPS and how many insecure redirects that occur before the website itself is returned. A website is considered loaded when the server returns a response code which does not indicate a redirect and considered as using HTTPS when the final page is loaded uses HTTPS. While any redirect to HTTPS is more secure than simply using HTTP, every insecure plaintext HTTP request sent while following redirects provide an additional opportunity for an attacker to modify or monitor the data sent using a Man-in-the-middle attack, possibly even preventing the redirect to HTTPS completely(see 2.4).

HTTP Strict Transport Security enables the server to tell the browser to never try to load a hostname using HTTP, only HTTPS (see 2.5.1), eliminating the potential initial insecure request. The scanner checks for the presence of this header, and attempts to parse it and extract the lifetime and whether it's preloaded or includes subdomains if it's present [53]. The raw header value is also saved for possible future analysis.

This data is collected by sending a HTTP request to the website and following redirects until we hit a max limit or get a non-redirect result. We then store the security of the final page and count the number of insecure redirects, as specified above.

3.2 TLS versions

First of all, the scanner collects which TLS versions are supported by a host, ranging from TLS 1.0 to TLS 1.3. SSL 3.0 and lower are excluded as their support is disabled in the OpenSSL versions distributed with several Linux distros, and according to Qualys SSL Pulse less than 300 of the top one 150 000 websites support these older versions (per 11th of April 2021) [44].

Knowing what versions of TLS are supported can provide valuable information about how quickly new major updates are rolled out. As with most software, bugs and vulnerabilities are sometimes discovered in TLS or TLS implementations, and the versions supported can give an indication about whether individual hosts are vulnerable to these. Some vulnerabilities are tied to specific TLS versions, while others might be tied to a software implementation version which only supports certain TLS versions. We can then derive information about whether a server might be vulnerable without needing to attempt to exploit the vulnerability itself, which can sometimes be problematic to do at scale and non-destructively.

This data is collected by attempting to create a connection to the host with each TLS version, only allowing one version at a time, then observing the result. If we successfully connect, the version will be listed as supported.

While listing supported ciphers would be useful, the scanner does not currently collect this information. A major reason for this is that many insecure cipher suites are not included in modern OpenSSL versions, and must be enabled with a flag during compilation for them to be available. This means that a scanner using a normal system installation of OpenSSL would be unable to scan for these less secure cipher suites, severely diminishing the utility and accuracy of the information.

3.3 Certificates

The scanner also collects data relating to the TLS certificate used by the host, relating to the issuing of the certificate, certificate transparency, and which authority issued it.

The scanner collects the certificate authority organisation name and common name, certificate type, serial, and whether the certificate is valid. To start with, the scanner opens a TLS connection using the most recent available version, and then check it's validity, save the certificate authority, certificate type, and serial. Additionally, the raw certificate is saved to enable future analysis.

It also stores whether an OCSP response is sent along with information about certificate transparency timestamps sent by the server. It does this by creating a TLS connection and observing the data logged about the handshake. Specifically, it checks if an OCSP response is sent, as well as the id and name of each certificate transparency timestamp. It also stores the raw timestamps for future analysis.

Last but not least, the scanner checks for the presence of a certificate

authority authorization DNS entry on the domain name. If this entry is present, the host is considered to be using certificate authority authorization, regardless of how restrictive or lenient it is.

Chapter 4

The scanning process

The scanning process itself is relatively simple, primarily involving opening and closing several TLS connection to the host with different settings and observing the results. In this chapter I will detail the order and relationship between how the different data are collected, such as when the scanning of some data depends on the scanning of other data.

The first part of the scan is to send a HTTP request to the server, and observe the response redirects and headers as specified in section 3.1. I also store whether we are able to connect at all, as several later parts of the scan will not produce any meaningful result if the scanner is unable to connect in the first place.

If the scanner was able to connect to the host in the beginning, it then starts making TLS connections to determine which TLS versions are supported (see 3.2), ranging from version 1.0 to version 1.3.

After this, the scanner will also check for the presence of a certificate authority authorization DNS record using a DNS query (see 3.3). This will be done even if the scanner was unable to establish any connection to the server.

If the scanner was able to connect, it then starts checking the certificate and the security features used during the handshake (see 3.3), including checking certificate validity, collecting certificate data, checking for an OCSP response, and collecting certificate transparency log timestamps.

Chapter 5

The implementation

This implementation has two major parts: a scanner and an analysis API. The scanner collects data and stores all the resulting data in a database for later analysis (see 6.3). The API then provides a simpler interface to export and query the data for further analysis (see 7.2).

The scanner part runs a scan against a list of hostnames. This list is loaded from a CSV file and then separated into one smaller list for each process used during the scan. Each process scans it's own list, as specified in chapter 4, and then saves the result for each hostname in the database after finishing the scan for that hostname, in a key-value based database schema.

The analysis API can then be used to query or export the data from the database, making the data available using a HTTP REST API. This can be done using SQL, or by exporting a list of fields for use in scripts or other software (see 7.2). This makes it fairly easy to access the data, as HTTP clients and json libraries are available to most programming languages.

5.1 Technologies used

In this section I will describe the technology choices I have made when creating the two parts of the project.

5.1.1 OpenSSL

OpenSSL is an library and command line tool to create, use, and manage encrypted SSL and TLS connections for other software [37]. It's distributed with several different Linux distributions, which means that other applications don't need to implement or include TLS implementations themselves. Instead they can use the system distribution, which can be kept up to date independently of the software itself. There are also other TLS libraries with similar functionality, for example BoringSSL [6], GnuTLS [14], and LibreSSL [31].

For this project, I have chosen to work with OpenSSL. This is because it is included with several Linux distributions, making it easy to access and

use. It also makes it a natural default for many Linux based systems and servers.

5.1.2 Python

Python is a scripting language, which I have chosen to use for this project along with several libraries. I will mention some of the libraries in this section. It's primary role during the scan is to call the right functions in the Python SSL library and OpenSSL, interpreting the resulting data, then saving the data in the database. To aid with analysis, I have also created a REST api providing access to query and export raw data.

I use the Python standard library TLS library when it provides enough documented flexibility. This then uses OpenSSL to handle the TLS connections [52]. However, many TLS options offered by OpenSSL are either not available here, or are not documented as available as most implementers should probably not override the defaults. To avoid potential future issues, I have chosen to only use the standard library when the functionality and options I need are documented.

When I need more control than the standard library offers, I have used pwntools [13] to call the OpenSSL command line interface [38]. The reason I use the CLI rather than a different set of OpenSSL bindings for Python is that many of the up to date bindings have the same issues as the standard library. You're expected to use default options, and customizing some options is either not documented or not made available.

For database access and queries, I use SQLAlchemy [5]. The query builder makes it simple to create flexible queries for the export endpoint filters, and it makes it easy to insert the results data during the scan.

5.1.3 PostgreSQL

PostgreSQL is an open source relational database management system, which is used to store and query the data collected by the scanner using SQL. Dating back to 1986 [41], it's a stable and efficient database management system which I also have prior experience with. This makes it an easy choice for the backend storage of this project.

Database design

The main goals of the database design is to be easy to extend and easy to query. To that end, I decided on a key-value based table schema with a view which combines the data to make it simpler to query. The design also allows some duplicate storage of values, which can help with performance during the scan as it won't need to look up any data in other tables in order to reference it properly in other tables.

The database uses a key-value style schema in order to be more extendable and to be queryable without needing knowledge of the JSON functionality in PostgreSQL [1]. The schema means that anyone can add any key to the database, without needing to modify the database itself. It

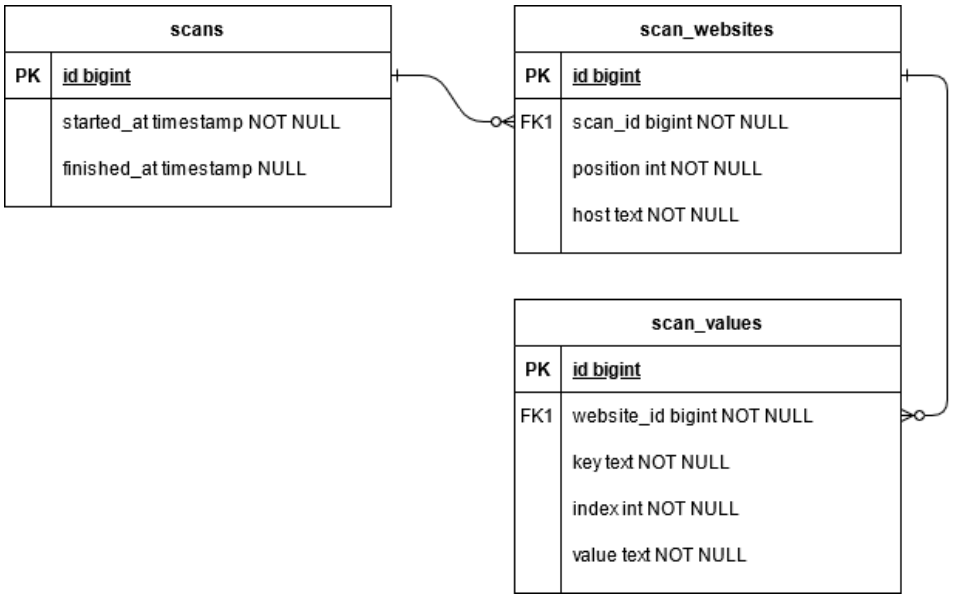

Figure 5.1: The database table schema. Data view is exluded.

also means that it's easy to add indexes and optimizations to the tables, in a normal SQL database fashion.

I have also created a query/data view, as an additional aide to users of the database. By gathering all the information in a single, queryable view, I have eliminated the need for most join operations and provided one unified interface which is also accessible through the query api. The view simplifies and removes some of the barriers by providing a powerful interface to the data.

Here is the database schemas used by the scanner and analyser.

Table 5.1: Table: scans
This table keeps track of the scans themselves. Every time someone runs a scan, it will be a new entry.

Column	Data type	Purpose
id	int64	Primary key
started_at	timestamp	When the scan was started.
finished_at	timestamp	When the scan was finished.

Table 5.2: Table: scan_websites
This is the list of websites processed in a scan. It stores the hostname, and the position on the list.

Column	Data type	Purpose
id	int32	Primary key
position	int64	The ranking of the website from the list.
host	string	The host name of the website. For example, uio.no.
scan_id	int64	References which scan this website entry was added by.

Table 5.3: Table: scan_values
This stores keys and values containing the scan results for each website.

Column	Data type	Purpose
id	int64	Primary key
website_id	int64	References which website this key/value belongs to.
key	string	Identifies what is stored in this entry. This is what identifies what part of the scan generated this value.
index	int32	In cases where a key has multiple values, this identifies which value this is for the purpose of processing related values from different keys.
value	string	The value for the specific key and index. This is where the result of a part of the scan is stored.

Table 5.4: View: data
This is a view which combines the data from each table, in order to simplify querying.

Column	Data type	Purpose
scan_id	int64	The id of the scan this data entry is from.
scan_start	timestamp	When the scan was started.
scan_finish	timestamp	When the scan was finished.
position	int32	The position of the website the entry belongs to.
host	string	The hostname of the website the entry belongs to.
key	string	The key which identifies the value the entry is storing.
index	int32	The index which identifies which instance of a key this is.
value	string	The value for this key and index.

Chapter 6

Running a scanner instance

In this chapter I will describe how one can set up and run an instance of the security scanner.

6.1 Installing the necessary software

First, set up a server or virtual machine running Ubuntu 20.04 LTS. Other versions and distros might work, but the development and testing of the scanner has been done using Ubuntu 20.04. The instructions will be written for Ubuntu 20.04.

As the scanner is built on top of OpenSSL and tested with OpenSSL 1.1.1f, the server or virtual machine have this installed. This is pre-installed with Ubuntu 20.04.

The scanner needs a PostgreSQL server, with two user accounts. The PostgreSQL 12 server can simply be installed using apt. After installing the server, create two user accounts and a database. One user should have write access and be able to create tables and views, while the other should only be given read-only access to the view named *data*, which will be created the first time the software itself is started. Make note of the database name, usernames, and passwords used. A PostgreSQL database server should now be up and running.

Next, install Python 3.8, Poetry, and the libraries the scanner depends on. Python 3.8 can be installed using APT (Advanced Package Tool), and afterwards Poetry [40] can be installed using PIP3. With Poetry installed, the Python dependenceis of the scanner can be installed by using the `poetry install` command. After this, `poetry shell` can be used in the project folder to start a shell with the correct Python environment loaded at any time.

6.2 Configuration

This section will explain the configuration file, as well as how to set it up. To start with, make a copy of *config.example.py* and call it *config.py*. After

this, the new *config.py* file can be opened and the information in it can be edited.

`DATABASE_URL` is the url used to access the main read/write database account. This account should have both read and write access to the database, and be able to create the database schema.

`DATABASE_READ_URL` is the read-only account specified in the previous section (6.1). This account only needs to have read access to the *data* view, which is created by the application the first time it's started.

`THREAD_COUNT` is the number of processes (not threads) used by the scanner to perform the scan. This can usually be quite high, as a large portion of the time each process spends scanning a website is simply it waiting for a response, rather than it doing any heavy processing.

`QUERY_TOKEN` is the API token used by the */api/query* endpoint, which allows anyone with the token to execute parameterized SQL without needing a PostgreSQL client library. This token has to be provided with every request to this endpoint, as even a read-only database account can do damage. The endpoint makes it easier to execute own queries in order to further analyse the data.

6.3 Running the scan

Before starting, several of these steps require using the correct Python virtual environment, created using Poetry. This can be done by using `poetry shell` in the project directory, which will start a shell with the correct environment.

First load the custom OpenSSL configuration. This can be done by typing `source openssl/set_conf.sh` in the project root directory, which will set an environment variable telling OpenSSL to load a configuration file which accepts weaker settings than it would accept by default. This is necessary for some of the scans to work correctly.

Second, the scanner needs a OpenSSL compatible list of trusted certificate transparency log servers, and it should be saved as *openssl/ct_logs.cnf* in the project directory. Fortunately, lists of trustworthy CT log servers used by broswers are easy to find and a script is included to fetch and convert Google Chrome's list. Simply run `python3 ct\_generator.py` in the project directory, and it will generate and save the file in the correct location.

Third, list of hostnames to scan is needed. This list should be a CSV file with a numerical ranking in column one and the domain name in column two, without a header row. This is compatible with several lists of the top websites, such as Tranco's list of the top 1 000 000 websites (see 2.6). This list can be saved anywhere, as long as it is readable by the scanner.

Finally, it's time to run the scanner itself. Execute `python3 scan_list.py [list path] <Resum` where list path is the path of the CSV file containing the host list. If a resume scan id is included the scanner will check each hostname against the database before scanning, in order to only scan previously missed or ignored websites. The scan might take a while and will save it's results in the PostgreSQL database created earlier while it is working. If a connection

issue occurs between the scanner and the database, the scanner will back off and try again several times before potentially discarding the data. Any unexpected errors will be saved in a folder named *errors/* in the working directory.

6.4 Serving the data

As with the scanner itself, these steps require using the correct Python virtual environment. This can be done by using `poetry shell` in the project directory.

For local and personal use, the built in development WSGI server (Web Server Gateway Interface server [12]) may be suitable. It can be started by using the command `python3 serve_dev.py`. Note that this server is not made to be exposed to the internet, and is only recommended for personal and development use cases.

To make it available to others, using a more robust WSGI server is recommended. One possible option for this is Gunicorn.

Chapter 7

Expanding the scanner and analysis

In this chapter I will explain how to expand on the scanner itself, as well as how to do further analysis on the collected data.

7.1 Viewing scan report

The scanner provides reports for every completed scan in the database. Simply navigate to *http://hostname/* (where hostname is the host of the scanner), and it will redirect the browser to the latest scan report. Previous scan reports can be accessed using the arrows at the top.

Each scan report shows statistical overview of the following:

- The support for each TLS version.
- The use of various certificate types (Domain validation, organisation validation, and extended validation).
- The number of websites redirecting from HTTP to HTTPS.
- The number of insecure (HTTP) redirects before a website is served over HTTPS.
- The use of HSTS, and the durations set.
- The use of HSTS preloading.
- The use of CAA.
- The number of certificate timestamps returned by the server.
- The use of OCSP stapling.

7.2 Accessing the REST API

The project offers a REST API to enable simpler querying and exports, usable from any language with a HTTP client and a json library. This makes analysing the data possible in most language, as most languages have libraries providing a HTTP client and json support. For information on how to run the API, see section 6.4.

The API is documented in accordance with the OpenAPI Specification [34], and can be viewed using Swagger UI. The simple way of doing this when running the API is to access *http://hostname/docs* (where hostname is the host of the scanner), which will redirect to Swagger UI bundled with the application. Swagger UI will display all available endpoints and allows sending test requests to the API without leaving the page. The raw specification file is also accessible at at *http://hostname/api/spec*, which can then be used to generate an API client or import into other tools.

/api/keys gives access to a simple list of available keys, returned as a string array. These are the keys which can be used when querying or exporting data, based on the existing data in the database. This means that prior to the first scan, this list will be empty. At the same time, if the scanner is expanded to collect more data, this endpoint will include the key automatically.

/api/scans returns a list of all scans in the database. This list includes the start time, finish time, and the id of the scan. If the scan is incomplete, the finish time will be *null*. This can be used to determine which scans are in a relevant time frame for an analysis and then use the corresponding scan ids to efficiently limit a query or an export to only these scans.

/api/export allows export a list of keys for a list of scans. The requested data is returned as a list of datapoints, each containing the requested keys for a specific website in a specific scan along with relevant metadata. This metadata is stored as keys beginning with an underscore (_) and includes values such as the scan id, the website hostname, it's ranking in this scan, and the scan start and finish time. The results for the requested keys are returned as lists in the object, in order to handle cases where one key has multiple values attached to it. When multiple keys are used to store multiple values about the same element, for exampel for certificate transparency timestamps, the values belonging to the same element will always have the same index. For example, the CT log name with index 1 always belongs to the same timestamp as the CT log id with index 1.

/api/query allows the execution a raw SQL query against the data view (see 5.1.3 for the view schema), returning a list of the resulting rows from the database. As this endpoint allows some code execution, it is protected by a shared secret token which is specified in the configuration file. This shared token is required to run queries. The endpoint supports parameterized queries, which can be utilized by simply adding a parameter such as ":scanid" to the query along with a corresponding parameter name and value to the json blob in the request. This endpoint thus allows for more advanced querying without writing any code outside of an SQL statement, and without needing to find and install a PostgreSQL

database driver.

More information about these endpoints can be found in the OpenAPI specification and using Swagger UI, as mentioned previously.

7.3 Scanning for other data

Scanning for additional data is a simple two-step process, editing the Python script slightly.

To start with, write the code to find and save the additional data. First, open *masterscanner/website.py* and create a method in the *Website* class. This method should collect the necessary information and store the result in the *results* dictionary. Any value other than *None* in this dictionary will be saved automatically, and made available to queries and exports. The hostname being scanned can be accessed in any method on the *Website* class by accessing the *host* variable on the class. The method should handle most exceptions which might occur during a scan, any uncaught exceptions will be logged to the *errors* folder in the project directory and cause the scanner to ignore the data collected from the website, as it might be partially incorrect due to the unhandled exception.

Afterwards, a method call to the new method needs to be added for it to be executed. In order to include a new method when scanning a list, open the *scan_list.py* file and add a call to the method in the *do_scan* method, along with the other scan method calls. This will make the scanner call the method during the list scan, which will in turn cause the scanner to save any data you add to the result dictionary for further analysis.

Finally, while not required, adding some automated tests to ensure your addition is correct is recommended. The test suite is managed by pytest, and all tests are located in the *tests* folder. To add tests for an addition, create a python file with a name ending with "_test.py" in this folder. In this file, define methods with names starting with "test_" with test cases for the addition. Call the added method, then use *assert* statements to check that the data added by the method is correct. Run the `pytest` command in the Poetry shell to execute the test suite (see 6.1).

Chapter 8

Evaluation

My evaluation of the implementation will be done in three main parts. The first part will comment on the use of automated testing to verify the correctness of the scan. The second part will compare my scanner's results to the results of other similar scanners and comment on this. The third part will evaluate it's performance and

The primary qualitative check of correctness for my scanner is through automated testing. These tests execute portions of a scan against specific websites with specific configurations in order to ensure that the scan of each website gives the expected result. This is done for all the data collected by the scanner, and it should provide a high degree of confidence in the scanner itself assuming the tests are correct.

The second check of correctness is a comparison of this scanners results with those of other scanners. While different sample sizes, lists and implementations makes it difficult to compare the numbers exactly, major unexplainable differences in the resulting data can indicate that parts of the data is incorrect.

8.1 Automated testing

As mentioned, this scanner extensively uses automated testing, comparing the results from using the scanner against various websites with those of other tools. These tests are ran against websites with specific configurations, for example various of the test-websites operated by badssl.com and other websites with configurations I wish to test against. This also has a downside, however, as those 3rd party websites may change their configuration and thus cause the test to fail despite the scanner implementation being correct. To prevent this, the tests require regular maintenance and updates.

These tests have played an important role in the development of the tool. By analyzing a website using other tools, then automatically running my own scanner and comparing the results I have been able to develop and test the scanner in an efficient manner, as well as ensure that future changes and improvements I have made have not impacted the results of the existing parts of the scanner.

8.2 Comparing the results to other scanners

In this part of my evaluation I will compare the results of a scan with this implementation with two other sources, Qualys SSL Labs SSL Pulse [44] and Scott Helme's Crawler.Ninja [17]. This scanner will be using the top 10 000 websites from the Tranco list [30], while Crawler.Ninja uses the full top 1 million from the Tranco list [20], and SSL Pulse uses a list of 150 000 large websites based on the Alexa top 1 million websites list [44]. I will present the data, then compare and comment on it.

While this method of evaluation might expose some obvious mistakes, it also has a weakness caused by the scanners using different sample sizes and lists. In some cases I can compare my scan with the top 10 000 website results from another scanner, but this is often not possible. As such, differences must be

The Qualys SSL Labs SSL Pulse data is from a scan done on the 4th of May 2021, the Crawler.Ninja data is from a scan done the 9th of May 2021, and this scanner was run on the 10th of May 2021 based on the Tranco list from the 9th of May 2021. The percentages for this scanner and Crawler.Ninja are based on the number of websites reached by the scanner, rather than the total number of websites the scanner attempted to connect to.

Table 8.1: Scan results, comparing to Qualys SSL Labs SSL Pulse (4th of May 2021) [44]

Data point	Count	% of available websites	SSL Pulse %
Websites available	8731	-	-
Uses HSTS	3669	42.0%	28.5%
Uses CAA	1225	14.0%	8.7%
Uses EV certificate	397	4.5%	6.1%
Uses OCSP stapling	3417	39.1%	40.8%
Supports TLSv1.0	4186	47.9%	46.4%
Supports TLSv1.1	4641	53.2%	51.6%
Supports TLSv1.2	8300	95.1%	99.5%
Supports TLSv1.3	4020	46.0%	45.3%

Table 8.2: Scan results, comparing to Crawler.Ninja (10th of May 2021) [17]

Data point	Count	% of available websites	Crawler.Ninja count	Crawler.Ninja %
Websites available	8731	-	834017	-
Redirects to HTTPS	7829	89.7%	572491	68.6%
Uses HSTS	3669	42.0%	167731	20.1%
Uses CAA	1525	17.5%	26793	3.2%
Uses EV Certificate	397	4.5%	11337	1.4%

Table 8.3: Scan results, combined data

Data point	Scanner %	SSL Pulse %	Crawler.Ninja %
Websites available (count)	8731	150000	834017
Redirects to HTTPS	89.7%	-	68.6%
Uses HSTS	42.0%	28.5%	20.1%
Uses CAA	17.5%	8.7%	3.2%
Uses EV certificate	4.5%	6.1%	1.5%
Uses OCSP stapling	39.1%	40.8%	-
Supports TLSv1.0	47.9%	46.4%	-
Supports TLSv1.1	53.2%	51.6%	-
Supports TLSv1.2	95.1%	99.5%	-
Supports TLSv1.3	46.0%	45.3%	-

When looking at the result of the entire scan, there is a notable difference between this scanner and Crawler.Ninja on the percentage of websites which redirect to HTTPS. However, if I did a bit more into the data and limit the Crawler.Ninja data to only the top 10 000 ranked websites, we get a very different picture. Out of the top 10 000 ranked websites 7326 websites redirect to HTTPS, or about 73% of the total list. If I assume that Crawler.Ninja managed to reach the same number of websites as my scanner, this increases to 84%. Some of the difference might be due to transient issues such as application errors, but not all of it. The majority of the difference is likely caused by differences in how redirects are followed, for example by this scanner supporting a large variety of redirect codes (including less used ones), including 301, 302, 303, 307, 308 [45]. Ultimately, I consider this scanner to be a reliable indicator in this regard, as the redirects are handled by an underlying library which supports a large variety of the available HTTP redirect status codes beyond the most used ones.

Looking at the HSTS usage figures, the scanners all report different percentages. This likely comes down to the differences in sample size and list scanned. However, if I once again check the Crawler.ninja data and only include the top 10 000 rank websites I get that 3457 websites use the HSTS header, compared to the 3669 websites reported by my scanner. Those numbers are quite close which supports my suspicion that the differences when including the entire dataset comes from the different lists. It's hard to come with any conclusions about the difference between the top 10 000 from my scanner and from Crawler.Ninja, but it's likely that Crawler.Ninja considers websites with a HSTS duration below a certain thresholds as having HSTS disabled while my scanner still considers it enabled. This is supported by a number of websites having unusually low HSTS duration in my results as opposed to the common 6 months or higher, though it is difficult to confirm due to it being closed source. Additionally, some websites serve different security headers depending on the connecting

client. This means that my scanner, using a Mozilla Firefox user agent, might get different results from a scanner which uses a Google Chrome user agent.

Regarding the use of OCSP stapling and the support of various TLS versions, the differences between my scanner and SSL Pulse are quite small with the largest difference being 4.4 percentage points in the support for TLSv1.2 figure. Once again, the most likely reason for this difference is the different sample sizes and lists used by the scanners.

Overall, while there are some differences between the different scanners, most seem to be easily explainable. Some can likely be attributed to the different sample sizes and lists, while others can likely be attributed to specific differences in how a scanner might interpret the data it collects. It's worth noting that it is difficult to prove this attributions, due to the closed source nature and more limited analysis options of the other scanners.

8.3 Performance and storage use

In this part of my evaluation I will look at the performance of the scanner along with how much storage it uses per scan. In this benchmark the scanner was run on a cloud based virtual machine running Linux Ubuntu 20.04, with two shared virtual CPU cores and two gigabytes of memory available to it. The process count was set to 50 processes. The PostgreSQL database is hosted on the same virtual machine as the scanner.

Using this configuration the scan itself finished in about 45 minutes. During this time the CPU and memory was not a bottleneck for the scanner, instead most of the time was spent waiting for responses over the network and for operating system PTY pseudoterminals to be available [43]. The CPU usage peaked at 23.8% and the memory usage peaked at 76%. With more memory, the number of processes could likely be doubled with only a double core CPU.

When scanning the top 10 000 websites the scanner saved an average of about 6.29 KB of information per website, not including database indexes, adding up to 61.43 MB of data. As shown in figure 8.1 this adds up to about 614.26 MB for 100 000 websites, or 6142.58 MB for 1 000 000 websites. In addition, there are the database indexes which add up to 12.55 MB for this first scan.

The frequency of scans and sample size used for scans should keep this size growth in mind. When running the scanner more often, it's worth considering using a smaller sample size or regularly deleting old data as it stops being relevant. For a daily scan one might want to use a sample size of less than 100 000, while for a monthly scan the increased size from scanning the entire Tranco top 1 000 000 list might be acceptable. The right frequency and sample size should consider the usefulness of more frequent scans and the amount of storage space available.

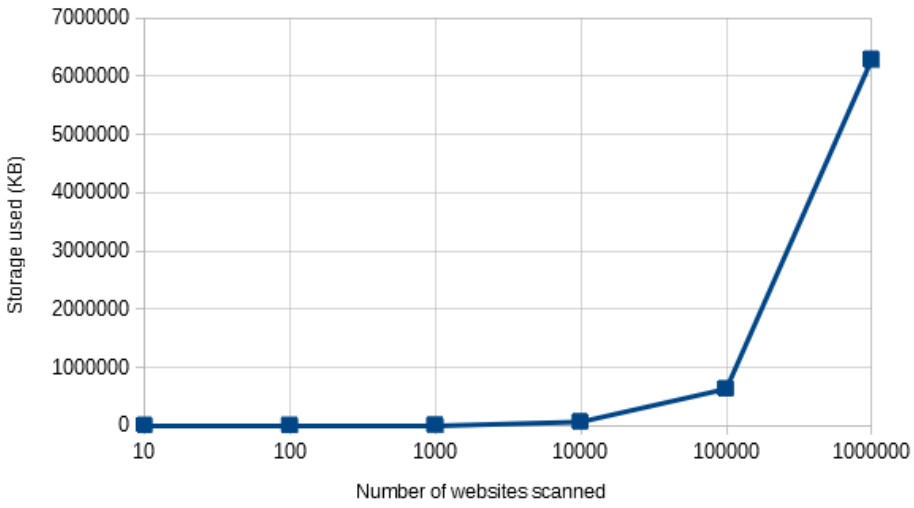

Figure 8.1: Storage use in KB for number of websites scanned.

Chapter 9

Conclusion

9.1 Summary

In this thesis, I have presented an extendable internet security scanner which enables both the collection of and further analysis of data regarding the TLS and HTTP security configuration of websites. This scanner can read and scan an entire list of websites, and will then save the results in a database. It can easily be extended to collect more information by doing small modifications to the code, to increase the utility of the scanner.

The project also has a web application, which serves scan reports and several json based REST api endpoints which allow for export of and querying data for further analysis. The exporting functionality allows the user to specify exactly which data they want to export and for which scans. The data can also be combined with other datasets from other sources, for example to create a larger collection of data about a website.

When combined, this provides an extendable scanner with easy access to data for further analysis. This can be valuable for monitoring the state of TLS use, and be an enabler for further research into the real world use of TLS.

9.2 Further development

A useful addition to the scanner would be to check for more common vulnerabilities. While we can draw some conclusions regarding vulnerabilities and exploits based on supported TLS versions, this is not the most reliable option. There are vulnerabilities which do not affect TLS as a protocol, but might instead affect specific TLS implementations. In these cases, having definite datapoints proving or disproving the presence of a vulnerability can be a very useful addition.

Many large websites serve user uploads from a separate domain name from their main website, sometimes using a content delivery network (CDN). As an example of this, Facebook usually serves user uploads from the domain fbcdn.net rather than directly from facebook.com [56]. In these cases, the apex of CDN domain might not even resolve to a website, meaning it won't be included by this scanner. A system which is able to

detect and handle these CDN domains can provide useful data about these domains.

The scanner does also not consider various HTTP based application programming interfaces (APIs) which might be used by the website. For example, single page applications will often heavily rely on background API calls in order to function. In these cases, the security of the API is as important as the security of the main website itself, and being able to detect and scan the APIs used in these background calls would be a good addition to the scanner.

Bibliography

[1] *9.15. JSON Functions and Operators*. en. Feb. 2021. URL: https://www.postgresql.org/docs/12/functions-json.html (visited on 12/05/2021).

[2] Eman Salem Alashwali and Kasper Rasmussen. 'What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS'. In: *arXiv:1809.05681 [cs]* (Jan. 2019). arXiv: 1809.05681. URL: http://arxiv.org/abs/1809.05681 (visited on 26/05/2020).

[3] *Alexa - Top sites*. URL: https://www.alexa.com/topsites (visited on 26/05/2020).

[4] Elaine Barker. *Recommendation for Key Management: Part 1 – General*. en. Tech. rep. NIST Special Publication (SP) 800-57 Part 1 Rev. 5. National Institute of Standards and Technology, May 2020. DOI: https://doi.org/10.6028/NIST.SP.800-57pt1r5. URL: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final (visited on 27/05/2020).

[5] Micheael Bayer. 'SQLAlchemy'. In: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012. URL: http://aosabook.org/en/sqlalchemy.html.

[6] *boringssl - Git at Google*. URL: https://boringssl.googlesource.com/boringssl/ (visited on 23/04/2021).

[7] *CA/Browser Forum Guidelines For The Issuance And Management Of Extended Validation Certificates*. URL: https://cabforum.org/extended-validation/ (visited on 22/05/2020).

[8] Franco Callegati, Walter Cerroni and Marco Ramilli. 'Man-in-the-Middle Attack to the HTTPS Protocol'. eng. In: *IEEE Security & Privacy* 7.1 (2009). Publisher: IEEE, pp. 78–81. ISSN: 1540-7993. DOI: 10.1109/MSP.2009.12.

[9] *Certificate Authority Authorization (CAA) - Let's Encrypt - Free SSL/TLS Certificates*. URL: https://letsencrypt.org/docs/caa/ (visited on 27/05/2020).

[10] *Certificate Transparency*. Library Catalog: www.certificate-transparency.org. URL: http://www.certificate-transparency.org/home (visited on 27/05/2020).

[11] Dave Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc5280 (visited on 21/05/2020).

[12] Phillip J. Eby. *PEP 333 – Python Web Server Gateway Interface v1.0*. en. URL: https://www.python.org/dev/peps/pep-0333/ (visited on 12/04/2021).

[13] *Getting Started — pwntools 4.3.1 documentation*. URL: https://pwntools.readthedocs.io/en/stable/intro.html (visited on 23/04/2021).

[14] *GnuTLS*. en. URL: https://www.gnutls.org/ (visited on 23/04/2021).

[15] Dan Goodin. *Russian-controlled telecom hijacks financial services' Internet traffic*. URL: https://arstechnica.com/information-technology/2017/04/russian-controlled-telecom-hijacks-financial-services-internet-traffic/ (visited on 14/05/2021).

[16] Scott Helme. *Are EV certificates worth the paper they're written on?* en. Library Catalog: scotthelme.co.uk. Dec. 2017. URL: https://scotthelme.co.uk/are-ev-certificates-worth-the-paper-theyre-written-on/ (visited on 22/05/2020).

[17] Scott Helme. *Crawler.Ninja*. URL: https://crawler.ninja/ (visited on 13/03/2020).

[18] Scott Helme. *OCSP Must-Staple*. en. Library Catalog: scotthelme.co.uk. Feb. 2017. URL: https://scotthelme.co.uk/ocsp-must-staple/ (visited on 27/05/2020).

[19] Scott Helme. *Revocation is broken*. en. Library Catalog: scotthelme.co.uk. July 2017. URL: https://scotthelme.co.uk/revocation-is-broken/ (visited on 27/05/2020).

[20] Scott Helme. *Top 1 Million Analysis - September 2019*. en. Library Catalog: scotthelme.co.uk. Oct. 2019. URL: https://scotthelme.co.uk/top-1-million-analysis-september-2019/ (visited on 27/05/2020).

[21] Jacob Hoffman-Andrews, Phillip Hallam-Baker and Rob Stradling. *DNS Certification Authority Authorization (CAA) Resource Record*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc8659 (visited on 27/05/2020).

[22] *HSTS Preload List Submission*. URL: https://hstspreload.org/ (visited on 27/05/2020).

[23] *HTTPS encryption on the web – Google Transparency Report*. URL: https://transparencyreport.google.com/https/overview?hl=en (visited on 12/05/2020).

[24] Troy Hunt. *Troy Hunt on Twitter*. URL: https://twitter.com/troyhunt/status/691166196268417024 (visited on 14/05/2021).

[25] Troy Hunt. *Understanding HTTP Strict Transport Security (HSTS) and preloading it into the browser*. en. Library Catalog: www.troyhunt.com. June 2015. URL: https://www.troyhunt.com/understanding-http-strict-transport/ (visited on 27/05/2020).

[26] *Issues - project-zero*. URL: https://bugs.chromium.org/p/project-zero/issues/list?q=&can=1 (visited on 19/05/2020).

[27] Collin Jackson, Adam Barth and Jeff Hodges. *HTTP Strict Transport Security (HSTS)*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc6797 (visited on 13/03/2020).

[28] Ryan Kearney. *ComcastInject.html*. URL: https://gist.github.com/ryankearney/4146814 (visited on 14/05/2021).

[29] Adam Langley, Emilia Kasper and Ben Laurie. *Certificate Transparency*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc6962 (visited on 27/05/2020).

[30] Victor Le Pochat et al. 'Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation'. en. In: *Proceedings 2019 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23386. URL: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_01B-3_LePochat_paper.pdf (visited on 13/05/2020).

[31] *LibreSSL*. URL: https://www.libressl.org/ (visited on 23/04/2021).

[32] Bill Marczak et al. *China's Great Cannon*. URL: https://citizenlab.ca/2015/04/chinas-great-cannon/ (visited on 14/05/2021).

[33] Eric Mill. *Eric Mill on Twitter*. URL: https://twitter.com/konklone/status/598696478018666496 (visited on 14/05/2021).

[34] Darrel Miller et al. *OpenAPI Specification 2.0*. URL: https://spec.openapis.org/oas/v2.0 (visited on 08/05/2021).

[35] Bodo Moeller and Adam Langley. *TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc7507 (visited on 29/05/2020).

[36] Patrick Howell O'Neill. *It's easy to fake Extended Validation certificates, research shows*. en. Library Catalog: www.cyberscoop.com Section: Technology. Dec. 2017. URL: https://www.cyberscoop.com/easy-fake-extended-validation-certificates-research-shows/ (visited on 22/05/2020).

[37] *OpenSSL*. URL: https://www.openssl.org/ (visited on 23/04/2021).

[38] *OpenSSL commands*. URL: https://www.openssl.org/docs/man1.1.1/man1/ (visited on 23/04/2021).

[39] Yngve Pettersen. *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc6961 (visited on 13/03/2020).

[40] *Poetry - Python dependency management and packaging made easy*. URL: https://python-poetry.org/ (visited on 26/03/2021).

[41] *PostgreSQL: About*. URL: https://www.postgresql.org/about/ (visited on 23/04/2021).

[42] Matthew Prince. *The Hidden Costs of Heartbleed*. en. Library Catalog: blog.cloudflare.com. Apr. 2014. URL: https://blog.cloudflare.com/the-hard-costs-of-heartbleed/ (visited on 27/05/2020).

[43] *pty(7) - Linux manual page*. URL: https://man7.org/linux/man-pages/man7/pty.7.html (visited on 12/05/2021).

[44] *Qualys SSL Labs - SSL Pulse*. URL: https://www.ssllabs.com/ssl-pulse/ (visited on 13/03/2020).

[45] *Redirections in HTTP - HTTP | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections (visited on 30/04/2021).

[46] Julian F. Reschke and Roy T. Fielding. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc7230 (visited on 19/05/2020).

[47] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc5246 (visited on 11/03/2020).

[48] *Revocation checking and Chrome's CRL*. URL: https://www.imperialviolet.org/2012/02/05/crlsets.html (visited on 27/05/2020).

[49] S Santesson et al. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. en. Library Catalog: tools.ietf.org. URL: https://tools.ietf.org/html/rfc6960 (visited on 27/05/2020).

[50] Michael Satran et al. *Cipher Suites in TLS/SSL (Schannel SSP) - Win32 apps*. en-us. Library Catalog: docs.microsoft.com. URL: https://docs.microsoft.com/en-us/windows/win32/secauthn/cipher-suites-in-schannel (visited on 27/05/2020).

[51] Ryan Sleevi. *Announcement: Requiring Certificate Transparency in 2017 - Google Groups*. URL: https://groups.google.com/a/chromium.org/forum/#!searchin/ct-policy/Announcement$3A$20Requiring$20Certificate$20Transparency$20in$202017/ct-policy/78N3SMcqUGw/ykIwHXuqAQAJ (visited on 27/05/2020).

[52] *ssl — TLS/SSL wrapper for socket objects — Python 3.8.9 documentation*. URL: https://docs.python.org/3.8/library/ssl.html (visited on 23/04/2021).

[53] *Strict-Transport-Security*. en. Library Catalog: developer.mozilla.org. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security (visited on 27/05/2020).

[54] Nick Sullivan. *High-reliability OCSP stapling and why it matters*. en. Library Catalog: blog.cloudflare.com. July 2017. URL: https://blog.cloudflare.com/high-reliability-ocsp-stapling/ (visited on 27/05/2020).

[55] Justin Watt. *Hotel Wifi JavaScript Injection*. URL: https://justinsomnia.org/2012/04/hotel-wifi-javascript-injection/ (visited on 14/05/2021).

[56] Huapeng Zhou et al. *The Evolution of Advanced Caching in the Facebook CDN*. URL: https://research.fb.com/blog/2016/04/the-evolution-of-advanced-caching-in-the-facebook-cdn/ (visited on 14/05/2021).

Chapter 10

Appendixes

10.1 Appendix A: The source code

See the attached file named *masterscanner-master.zip*, or clone it from Github as humanewolf/masterscanner.