CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Large-scale Security Analysis of HTTP Responses

Utilizing Common Crawl to Scan the Web

Master's thesis in Computer science and engineering

JONATHAN CARBOL
HUGO STEGRELL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Large-scale Security Analysis of HTTP Responses

Utilizing Common Crawl to Scan the Web

JONATHAN CARBOL
HUGO STEGRELL

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Large-scale Security Analysis of HTTP Responses
Utilizing Common Crawl to Scan the Web
JONATHAN CARBOL
HUGO STEGRELL

Supervisor: Benjamin Eriksson, Department of Computer Science and Engineering
Advisors: Emilie Barse & Albin Eldstål-Ahrens, Assured AB
Examiner: Andrei Sabelfeld, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Word cloud of words used in this thesis.

Typeset in LaTeX
Gothenburg, Sweden 2023

Large-scale Security Analysis of HTTP Responses
Utilizing Common Crawl to Scan the Web
Jonathan Carbol
Hugo Stegrell
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Ensuring the security of computer systems has never been more critical as our reliability on software is ever-increasing. It is not enough to create systems that work securely during their development. Instead, systems and software need continuous development and updates to stay secure.

Inspecting individual web applications for security vulnerabilities and concerns gives a lot of feedback for the specific web applications being checked. However, the impact of particular vulnerabilities can be more accurately observed by looking for vulnerabilities in a larger sample size of web applications.

In order to find vulnerabilities on a large scale, this thesis utilizes the crawl data from Common Crawl, an open repository of web crawl data. In this data, indicators of specific programs, software, or libraries are visible, allowing us to find vulnerable versions. Utilizing cloud computing on AWS and the crawl data, this thesis showcases how to scan 3 billion websites to find indicators of vulnerabilities on millions of domains. Using dynamic verification of the results, thousands of these indicators of vulnerabilities are then shown to be verifiable.

Keywords: Security, cyber security, vulnerabilities, common crawl, AWS, cloud computing, computer science, engineering.

Acknowledgements

Throughout the project, we have faced struggles that would have been hard to overcome without the knowledge, feedback, and support of the people around us. We thank our supervisor Benjamin Eriksson for his ideas, support, and excitement. This thesis would not have been the same without you.

During the thesis, we also received support and expert knowledge from our supervisors at Assured AB. Thank you, Emilie Barse, Albin Eldstål-Ahrens, and the rest of the team at Assured for your invaluable expertise and help.

We also thank Andrei Sabelfeld for his support and feedback during our thesis. Finally, we would also like to thank Daniel Cronqvist and Saga Kortesaari for giving much-appreciated feedback on our report and writing.

Jonathan Carbol & Hugo Stegrell, Gothenburg, 2023-07-02

Contents

List of Figures

List of Tables

Glossary

API Application Programming Interface.

AWS Amazon Web Services.

CC Common Crawl.

CSRF Cross-Site Request Forgery.

CVE Common Vulnerabilities and Exposures.

EC2 Amazon Elastic Cloud Computing.

GHDB Google Hacking Database.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IOC Indicator Of Compromise.

JSON JavaScript Object Notation.

NIST National Institute of Standards and Technology.

OWASP Open Worldwide Application Security Project.

PHP PHP: Hypertext Preprocessor.

PoC Proof of Concept.

PT Path Traversal.

RCE Remote Code Execution.

RQ Research Question.

S3 Amazon Simple Storage Service.

SDK Software Development Kit.

SQL Structured Query Language.

SQLi SQL injection.

SQS Amazon Simple Queue System.

SSH Secure SHell Protocol.

TiB Tebibyte.

TLD Top Level Domain.

UI User Interface.

URL Uniform Resource Locator.

VPC Virtual Private Cloud.

WAF Web Application Firewall.

WARC Web ARChive.

WAT Web Archive Text.

WET WARC Encapsulated Text.

XSS Cross Site Scripting.

1

Introduction

The world is becoming more and more reliant on software and computers. Their impact on our daily lives has never been greater and it is therefore important to ensure the security of the software and computers we use every day. By increasing our security we can protect ourselves from malicious actors that seek to harm the confidentiality, integrity, and availability of the services and software we use.

Having security in mind when developing new applications or websites is very common. But developing something that at that moment seems secure does not necessarily mean that the application developed will always be secure. As new security flaws are detected, a program that was secure could become exploitable and vulnerable to different attacks. Therefore, it is important to review and update libraries, plugins, and other software used to ensure that the application stays secure.

Numerous programs and organizations track this kind of information, such as the CVE Program, which aims to identify and catalog publicly disclosed cybersecurity vulnerabilities [1]. Once these vulnerabilities have been discovered, there is still the task of identifying which ones impact oneself and how they can be fixed, and this kind of work is often neglected. An example of this is the Heartbleed vulnerability which was first discovered in 2014 and subsequently fixed. This vulnerability is believed to have allowed an attacker to steal millions of healthcare care records and was, in 2017, three years after having been patched, still present on over 200 000 devices [2].

This thesis demonstrates how one can help combat known vulnerabilities at a larger scale by analyzing data from web crawls and determining which classes of vulnerabilities are prominent. Common Crawl [3] is an open repository consisting of petabytes of web crawl data. By analyzing the data from Common Crawl with computing power through a distributed computing service, a vast amount of vulnerabilities can swiftly be detected and common vulnerabilities which have not yet been patched can be highlighted. It is also possible to identify websites that have already been attacked by finding indicators of compromise.

1.1 Goals

The thesis is divided into three research questions which will all have different goals and challenges associated with them.

RQ1 Can vulnerabilities be detected from the Common Crawl data?

RQ2 Is it possible to scan for vulnerabilities on a very large scale using the Common Crawl data?

RQ3 Can the found indicators of vulnerability be verified, e.g. by checking for false positives or running tests?

By showcasing examples of how vulnerabilities can be detected from the data supplied by Common Crawl, this thesis will demonstrate that utilizing crawl data to find vulnerable websites is possible. Furthermore, by using large-scale cloud solutions to find the vulnerabilities detectable in the Common Crawl data, as demonstrated, it can be proven that this methodology is valid on a larger scale. Finally, this thesis will verify the results found in the answers to previous research questions (**RQ1** and **RQ2**) to provide further validation of the methodology and the results obtained by it.

After answering all the research questions, this thesis can evaluate if it is possible to detect vulnerabilities on a large-scale utilizing HTTP responses from crawl data.

1.2 Contributions

This thesis will contribute to the computer science field in many ways. Firstly, we showcase techniques to detect services containing vulnerabilities from the Common Crawl data set. Secondly, we set an example of how large a data set can be processed to find the services with security vulnerabilities. Additionally, after gathering the results, we will evaluate them to prove their validity and give future work within the field a better understanding of how the process can be improved. Lastly, we compare our results to existing solutions, indicating the advantages and disadvantages of our method.

1.3 Limitations

As it is impossible to fit all known vulnerabilities into the scope of the thesis, a limitation must be imposed on the different types of vulnerabilities searched for. One example in each detection method type will be the first focus, to exemplify and illustrate the possibilities. After that, each detection method type can be extended to include multiple vulnerabilities. As Common Crawl provides crawling data from websites, it is impossible to detect other vulnerability types, such as hardware vulnerabilities or vulnerabilities in other types of applications, like SSH.

The data set that was used was the most recent snapshot from Common Crawl at the start of the thesis (January/February 2023 [4]). Although the data size of the Common Crawl archive is exceedingly large, it does not contain every web page that exists and also follows ethical crawling methods such as respecting common web crawler rules presented in `robot.txt` files. Due to that, some websites can avoid getting searched and thus reduce the data size available. Furthermore, as the data

is from a static snapshot and as the internet is ever-changing, the results found are only accurate for the instance in time during which the crawl was performed, and therefore, some vulnerabilities encountered may already have been patched or removed.

1.4 Related Work

There are several projects related to what is accomplished in this thesis. Some of these utilize the Common Crawl data to perform large-scale studies, whereas others perform similar scans on other data to find vulnerable systems and websites.

1.4.1 Large-Scale Data Processing with AWS Lambda

Several different studies have been done that search the Common Crawl archives for specific types of data. One such example is Chris Madden and Aaron Bawcom's work on *Analyzing Performance and Cost of Large-Scale Data Processing with AWS Lambda* [5]. In this study, they used the Common Crawl data set to find telephone numbers on websites. By utilizing thousands of AWS Lambda instances, they managed to search for strings that matched American phone numbers and thus manage to extract all these from the billions of web pages available. After seeing their success, we decided to try similar approaches utilizing Lambda and SQS to perform our large-scale scan. We however use multiple detection methods to scan for many different strings and then process the results further.

1.4.2 Shodan

Similar solutions to large-scale analysis of web application security already exist to some extent in the form of Shodan. Shodan is a freemium service and describes itself as a "search engine for Internet-connected devices" [6]. Since web applications are run on internet-connected devices, their information falls under what Shodan analyzes. By scanning servers, collecting information on all ports and services running, and storing this data, it can be used to find vulnerabilities on the web. With this data, Shodan wants to help provide data for *Network Security*, *Market Research*, *Cyber Risk*, *Internet of Things* and *Tracking Ransomware*.

While Shodan primarily focuses on finding vulnerabilities on all internet-connected devices, this thesis will try to showcase vulnerabilities found in web applications. Due to the closed-sources nature of Shodan, we can not know how exactly Shodan detects the vulnerabilities compared to the approach done in this thesis. By comparing the results obtained through our methods we can better understand how well our methods compare to the ones used by Shodan.

1.4.3 PublicWWW

Another similar service is PublicWWW, which describes itself as a source code search engine [7]. It stores data regarding code in web applications and allows users to

search for parts of this source code. Although PublicWWW's intended use is for marketing research, the service can still be used to find specific information related to security issues. The service can as such be useful as a measurement of how effectively our method fingerprints applications.

Searches can include anything from server versions, themes, and links, to scripting languages and the results display domains with the specified search terms. Currently, PublicWWW has scanned and indexed 477 865 948 websites from millions of domains [7]. We compare searches using PublicWWW and Shodan with the results obtained in this thesis in Section 6.5.

1.4.4 OWASP - Open Web Application Security Project

As the results from this thesis need to be validated, a dynamic analysis is required. This means that tests have to be created that check if the vulnerability, that is suspected to be active on a website, is actually vulnerable. In order to do these tests, information regarding how to perform them is required.

OWASP is a foundation that works to "improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences" [8]. They provide a guide called the Web Security Testing Guide that includes a framework and information regarding how to test web applications in the best way possible.

They also maintain an open-source web application scanner that automates the scanning procedure and helps detect vulnerabilities. Whilst it is not suitable for a large amount of data, such as the Common Crawl data set, it could be used by individuals trying to secure their own web applications.

1.5 Ethics

As with any thesis relating to cybersecurity, there are several ethical aspects to take into account. In order to ethically research the topic it is important that any malicious or destructive work is avoided at all cost. Not only does our own work need to be ethical, but any data released or disclosed by our work needs to be done responsibly by not naming any specific vulnerable pages or domains.

Whilst ethically, responsible disclosure of any detected vulnerabilities is always preferred, the scope and size of this thesis make it infeasible to disclose to vulnerable domain owners. This is further discussed in Section 7.4.

1.6 Thesis outline

Chapter 2 examines the background knowledge and theory required to understand the thesis. It details what kind of data is used in the thesis, what kind of vulnerabilities can be detected, and how web applications with the mentioned vulnerabilities can

be detected. The chapter also describes cloud computation services that can be used to create large-scale programs.

Chapter 3 describes the methods of fingerprinting vulnerabilities in HTML data. After that, it explains the contents of the file formats Common Crawl use and how the fingerprinting can be adapted to detect vulnerability indicators from those formats. This addresses RQ1, which is about the possibility of identifying vulnerabilities in the Common Crawl data.

Chapter 4 explains the implementation of a program that uses these methods as well as which vulnerabilities were searched for in the Common Crawl data. A proof of concept is created to test the solution. The program is then adapted and improved to work on a large scale with cloud computing, answering research question RQ2.

Chapter 5 explains the case studies performed for certain vulnerabilities to dynamically verify the results obtained by the scan. It goes into detail on how these vulnerabilities can be verified as well as why we need to verify the findings, hence describing how we can try to answer question RQ3.

Chapter 6 presents the results from the large-scale scan and detection of vulnerabilities and showcases the case studies performed for specific vulnerabilities to dynamically verify the results from the scan.

Chapter 7 discusses the results obtained from the scan as well as the dynamic analysis. It further discusses the false positives, ethical issues, and improvements that can be made to enhance the findings of this thesis. By analyzing the results from the scan as well as the dynamic verification it is possible to know for sure whether it is possible to detect vulnerabilities at a large scale as well as verify them and thus answer two of the research questions for this thesis (RQ2 and RQ3).

In **Chapter 8**, the conclusions are presented, along with suggestions for future work that can be performed to improve the results and gain more insights into vulnerabilities on the web, updated patterns, and tolerance for older versions used.

2

Background

This chapter presents the necessary background knowledge to understand the thesis. First, what Common Crawl is and how it works is described. Thereafter, the different categories of security vulnerabilities are presented, and how those can be identified from the data available in the Common Crawl archive and webpages directly. Lastly, the functionality that Amazon Web Services provide regarding distributed and serverless computing is explained.

2.1 Common Crawl

The data set used in this thesis comes from the Common Crawl project. Common Crawl maintains an open repository of crawl data, thus allowing for open access to internet information. This is done by crawling the web and collecting information such as metadata, HTTP headers, and raw HTML data.

Common Crawl ran its first crawl in 2008 and currently performs a crawl roughly every other month. Each crawl can be seen as a snapshot in time, and to this day they have performed 94 total crawls. Statistics for the crawl used in our analysis can be seen in Table 2.1.

Table 2.1: Common Crawl January/February 2023 archive statistics [9].

Websites crawled	3.15 billion
Hosts crawled	40 million
Domains crawled	33 million
Compressed data size	88 TiB
Uncompressed size	400 TiB

According to the Common Crawl engineer Sebastian Nagel, “Common Crawl is targeted to programmers, data scientists, researchers working with web data.” [10]. Their purpose is not to save the exact visual contents in their snapshots but instead to provide static data that can be statistically analyzed. Therefore, script contents and images are not recorded, meaning that full visual recreation of websites is not possible.

2.1.1 How the Crawler works

The crawler is a fork of Apache Nutch [11] and has been modified in order to easily create Web ARChive (WARC) records, change the algorithm for how it selects which websites to visit, and enable language detection of crawled websites. A crawl is performed with the help of Apache Hadoop in order to distribute the workload and gather the results.

As the crawler is built upon the crawling framework Apache Nutch, it utilizes a Nutch CrawlDb to select the URLs to be included in the next crawl [12]. The database is filled with URLs previously found as links on public pages or in sitemap files[1]. The amount of URLs sampled per domain depends on their harmonic centrality ranks, calculated on the latest hyperlink graphs.

The Common Crawl engineer Sebastian Nagel further described the selection of websites in the following way in 2018:

> "We also need to make sure that we provide representative samples in our "snapshot" crawls. That means we should include more pages from well-known, frequently visited domains, low-ranking domains are included with low probability. That's to somehow emulate the "random surfer" which will only occasionally visit a parked domain. And my guess would be that around 50% of the 300 million registered domains are parked. In the worst case, they host keyword and link spam which we try not to include (we cannot avoid it fully anyway) [13]."

Important to note is that the crawler follows policies in `robots.txt`. A website can therefore block the crawler by disallowing any robots or, specifically the user-agent 'CCBot'. This is as simple as including the following two rows in a `robots.txt` file hosted in the root of the website:

```
1 User-agent: CCBot
2 Disallow: /
```

2.1.2 Results from a Crawl

The crawler collects an extreme amount of data in Web ARChive (WARC) files. A WARC file contains the information gathered from multiple web pages in a manageable format (further described in Section 2.1.3).

All collected data for a scan is combined and compressed to form approximately 88 000 segments, each containing the data of around 36 000 websites. The dataset is published on Amazon S3 as part of the AWS Open Data Sponsorship Program[2] with each crawl in a separate directory. The crawls follow a naming scheme of `CC-MAIN-<year>-<week_of_year>` to easily distinguish when each snapshot was created.

[1] `https://sitemaps.org/`
[2] `https://aws.amazon.com/opendata/`

Each archive file is published as a gzip file. Using gzip 400 TiB of uncompressed data becomes 88 TiB (~22% of the original size). It also allows specific records to be extracted from the zipped format using offsets and the content length. Common Crawl also performs two conversions after each crawl (explained in Section 2.1.4 and Section 2.1.5) in case one has no interest in the raw response data.

2.1.3 WARC format

The Web Archive (WARC) file format allows multiple resources to be saved in a singular container file [14]. For each resource, the following information should exist within the WARC file:

- WARC metadata (version, date, type)
- The crawler HTTP request
- The website response to the HTTP request containing:
  - HTTP headers
  - Website metadata
  - Raw HTML data

Each resource is saved as a WARC record, starting with WARC record headers describing the contents, followed by a content block storing the raw data. An example of the file structure is shown in Figure 2.1. There are eight types of WARC records, and the header should specify the type of content. The type 'response' is used when saving the HTTP response from a web server and is the only type looked at within this project.

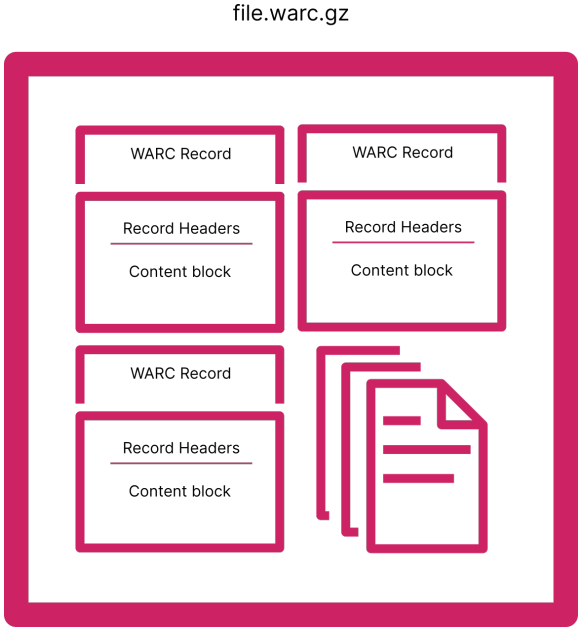

Figure 2.1: Structure of a compressed web archive file. There are an unknown amount of records until the file has been decompressed.

2.1.4 WAT format

Web Archive Transformation (WAT) is a conversion from the WARC format to one containing metadata [15]. Parts of the metadata that Common Crawl extracts are:

- HTTP headers,
- HTML data:
  - Meta tags
  - Page title
  - Links
  - Script tags

These are saved as a JSON object which is easily parsed to find the information of interest. One complete WAT record can be found in Appendix A, and an example of the schema for one of these WAT files can be seen in Listing 1.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "Container": {...},
    "Envelope": {
       "...",
        "Payload-Metadata": {
                        "...",
            "HTTP-Response-Metadata": {
                                "...",
                                "Headers": { },
                "HTML-Metadata": {
                    "Head": {
                        "Link": { },
                        "Metas": { },
                        "Scripts": { },
                        "Title": { }
                    },
                    "Links": { }
                },
                "Response-Message": {"..."}
            }
        },
        "WARC-Header-Length": { },
        "WARC-Header-Metadata": {"..."}
    }
}
```

Listing 1: The truncated JSON schema of WAT files.

2.1.5 WET format

The second conversion which Common Crawl provides, WARC Encapsulated Text (WET), is one containing the text of web pages. The format is also built using the WARC record format. Each record is prefaced with WARC metadata and then the extracted plaintext for that record. One complete WET record is shown in Appendix B. A minimized example from Common Crawl [15] of a conversion from the web page shown in Figure 2.2 to a WAT record is shown in Listing 2.

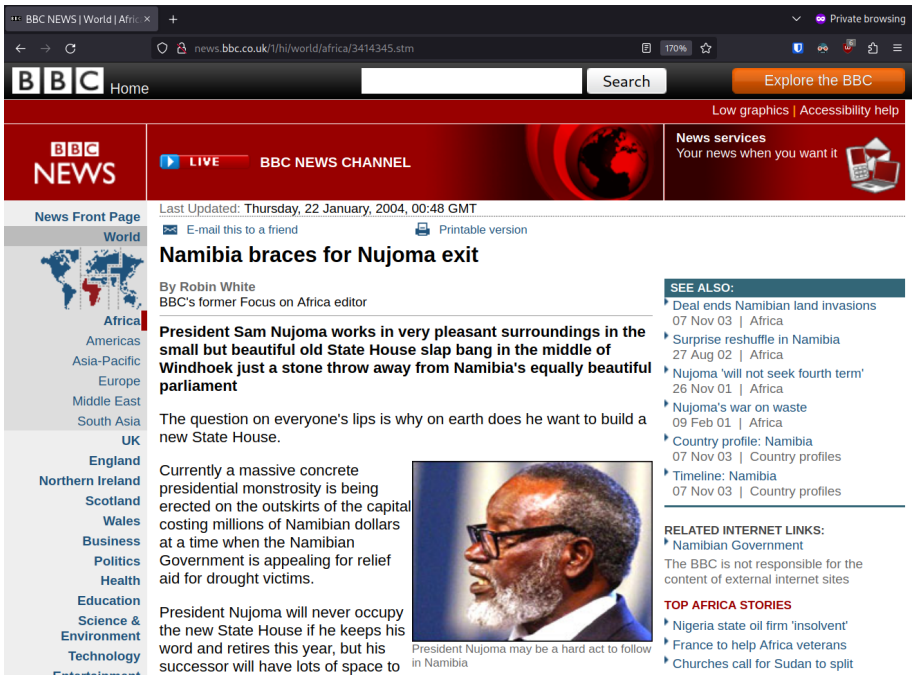

Figure 2.2: Screenshot of a BBC news article [16].

```
[WET Record Headers ...]

BBC NEWS | Africa | Namibia braces for Nujoma exit
[...]
Low graphics|Accessibility help
One-Minute World News
News services
Your news when you want it
News Front Page
Africa
[...]
-----------------
Video and Audio
-----------------
Programmes
Have Your Say
[...]
Namibia braces for Nujoma exit
```

```
By Robin White
BBC's former Focus on Africa editor
President Sam Nujoma works in very pleasant surroundings [...]
```

Listing 2: A truncated example of a WET file. Removed content is wrapped in brackets.

2.2 Vulnerabilities

There are endless amounts of known and unknown software vulnerabilities. A software vulnerability is defined as "A security flaw, glitch, or weakness found in software code that could be exploited by an attacker (threat source)." by NIST [17]. This section introduces vulnerabilities found in web applications but is constrained to more common categories which can be fingerprinted from Common Crawl data. Furthermore, we will briefly present the different attack surfaces within a web page that we will analyze, and for each vulnerability, we explain how the problem arises and how malicious users can abuse them.

2.2.1 Cross-Site Scripting

Cross-site scripting (XSS) vulnerabilities allow malicious scripts to be injected into benign or trusted websites [18]. These scripts run when another user accesses the web page or application as the browser assumes that they come from a trusted site.

Due to the user's browser not knowing that the script being run is malicious, the script can access cookies, session data, and other sensitive information. Additionally, the script can even rewrite the content displayed to the user. Figure 2.3 shows how an attacker may use stored cross-site scripting to steal users' cookies. There are also other types of XSS attacks, such as reflected and DOM-based, but we only present one type as the identification can be done in the same way.

A stored XSS attack, as in the example, means that the attack is persistent in some manner. In the figure, the attack is saved in a comment which executes the code once any user's browser renders that comment.

2.2.2 SQL injection

An application that is vulnerable to SQL injections (SQLi) allows a malicious attacker to inject their own query into an already existing SQL query [18]. This type of attack can extract information, delete items, drop database tables, and be extremely disruptive when performed. An SQLi attack could target queries that take user input that is not sanitized or parameterized. The SQL query below gets the name and address from a table of users where the ID of the row that should be retrieved is matched to an ID entered by a user.

```
SELECT name, address FROM users WHERE id=$id
```

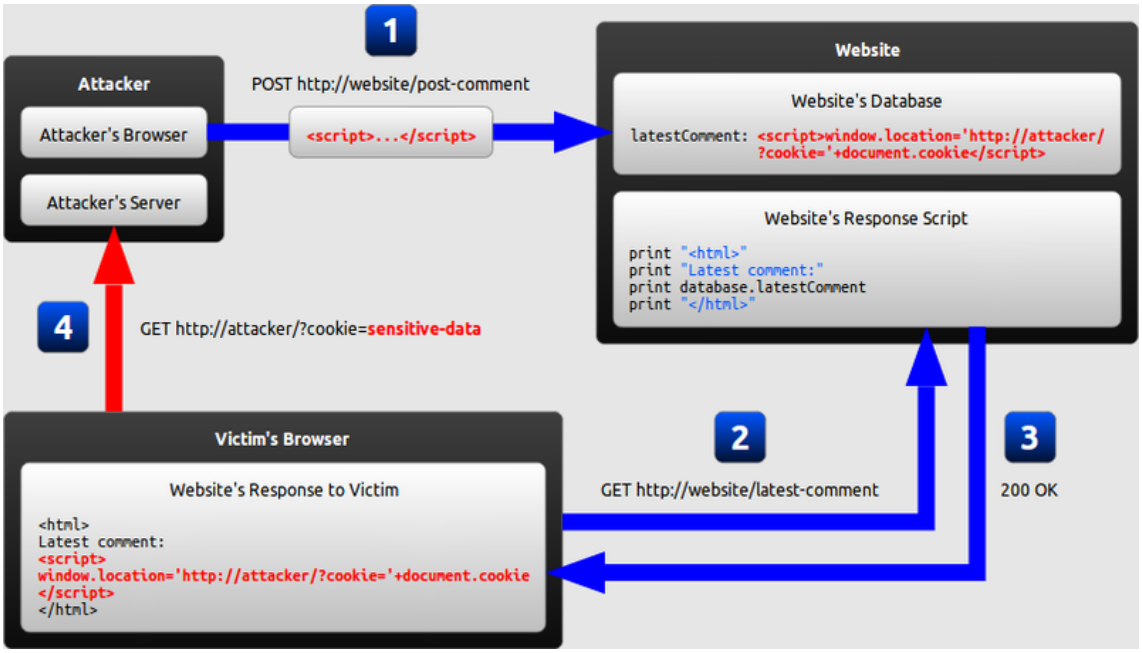

Figure 2.3: Example of a stored cross-site scripting attack. From [19]. CC-BY.

If a malicious user were to enter `'10 or 1=1'` the query being run would be:

```
SELECT name, address FROM users WHERE id=10 OR 1=1
```

This query would then get the name and address from the table users where the field `id = 10` or when `1=1` (which is always true), and thus all names and addresses would be returned, and not only the one matching to the ID.

2.2.3 Remote Code Execution

Remote code execution (RCE) or code injection is an attack that allows a nefarious user to execute their own code on the application [20]. If the nefarious user can exploit an RCE vulnerability, then they can execute whatever program they want and thus cause tremendous damage to the system being attacked. Anything from shutting down systems to extracting all data from the application could be possible if such a vulnerability is exploitable.

An example of an RCE attack would be a PHP application utilizing the `include()` function without input validation. This would allow the malicious attacker to utilize a URL that would normally pass a page name to the include function, instead pass a page from a malicious website instead. The below URL would represent the intended usage:

```
http://example.com/index.php?page=contact.php
```

Whereas the URL modified by the attacker would replace the `contact.php` page with a link to a malicious PHP script instead:

```
http://example.com/index.php?page=http://malicious.com/RCE_Attac⌋
↪ k.php
```

2.2.4 Prototype Pollution

Other than SQL injections, there are other types of injection attacks such as prototype pollution. Prototype pollution is an attack that utilizes the prototype object in JavaScript [21]. When getting access to the prototype setter for an object, the object being attacked can be mutated and is consequently vulnerable. By accessing the proto-object different kinds of attacks can be performed such as denial-of-service, privilege escalation, and more.

An example utilizing prototype pollution would be an attack on a vulnerable API that allows changing of parameters. By accessing the prototype setter, parameters that usually would not be accessible can be changed, and therefore the system can be attacked. Consider the following API which has a `POST` endpoint where a user's information can be updated:

```
POST -d '{"address": "Examplestreet 1"}'
↪  https://api.example.com/users/123

{name: "Emil Testsson", address: "Examplestreet 1", city;
↪  "Gothenburg", role: "user"}
```

The field role is inaccessible for the normal user, so a `POST` call to the endpoint trying to change it would not work:

```
POST -d '{"role": "admin"}' https://api.example.com/users/123

{name: "Emil Testsson", address: "Examplestreet 1", city;
↪  "Gothenburg", role: "user"}
```

However, if the prototype object were to be accessible due to a bad update or merge method, then the accessibility rules for the role field will be overridden by using the `__proto__` field.

```
POST -d '{"address": {"__proto__": {"role": "admin"}}}'
↪  https://api.example.com/users/123

{name: "Emil Testsson", address: "Examplestreet 1", city;
↪  "Gothenburg", role: "admin"}
```

As is seen in the example, the role was updated to admin even though the user should not be able to change it.

2.2.5 Path Traversal

As almost all web pages and applications need to include resources such as images, scripts, or themes, the problem of accessing these safely arises. While making certain resources publicly available is a necessity, other resources should not be reachable without the right privileges. Certain attacks utilize traversal in these open resource paths to find resources and files that should not be available [22]. This can allow a malicious user to obtain private and sensitive information.

Path traversal attacks can be of a lower complexity compared to other attacks that utilize custom scripts or payloads. The path traversal attacks can be done manually by using `../` to move upwards in directories. An example of a path traversal could be a website with links as seen below:

```
http://example.com/get-files.jsp?file=data.pdf
http://example.com/example-page.asp?page=index.html
```

These links either get files or other pages using the `get-files.jsp` function or by requesting a different page altogether. By changing what page or file is requested, unintended files can be disclosed:

```
http://example.com/get-files.jsp?file=/etc/passwd
http://example.com/example-page.asp?page=../../some dir/file
```

2.2.6 Arbitrary File Uploads

A web page may allow uploads of files; however, the files are not always validated to be of the correct file type or content. Depending on the services which the page is running, the uploaded file could then be used in an exploit and execute commands or even open a backdoor to the server [23].

Say, for instance, that a web page allows users to upload images which can later be viewed by navigating to the path `/uploads/{{filename.ext}}`. If the file is not validated correctly, an attacker may upload a web shell or malicious script. The web shell/file can then be navigated to and used with parameters in order to get system access or perform other malicious actions. A specific example of this would be uploading a `.jsp` file into the web tree. This file could be executed when navigated to and could include a malicious script that is subsequently executed [24].

2.2.7 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) enables an attacker to execute actions on another end user's web application session [18]. These attacks are often executed by tricking the end user into submitting malicious requests. In the case that a normal end user is compromised, the attack vectors can be login information leakage, transferring of funds, or altering the user's settings. If the user has administrative privileges, the entire web application may be compromised.

The attacks will only have an effect if the end user is already authenticated on the web page and if the application accepts any well-formed HTTP requests. The following example illustrates a payload that may be used in a CSRF attack that originates from one website but affects another. This attack utilizes the session data to switch an account's email address and as such provide the attacker with access to the now altered account.

```
<script>
function exampleAttack () {
    form.email = "attacker@example.com";
```

```
    form.submit();
}
</script>
<body onload="javascript:exampleAttack();">
<form action="http://example.com/updateAccount.php" id="form"
↪  method="post">
    <input type="hidden" name="firstname" value="Funny">
    <input type="hidden" name="email">
</form>
```

2.2.8 Privilege Escalation

Suppose a web application is vulnerable to privilege escalation. In that case, a user can access more information and perform more tasks than they usually should be allowed to do [25]. This can occur by allowing users to set their role to that of a higher privilege or by masking that they are an admin-type user.

There are two common types of privilege escalation: denoted horizontal and vertical escalation. Horizontal escalation occurs when a user is able to access resources of another account but of the same privilege level. In contrast, vertical escalation indicates a user gaining the privileges of a higher role than they currently are (usually that of an admin account).

The vulnerability could be something as simple as a site with different options in the menus depending on the role of the user, where an Admin would have access to `DELETE`, `ADD`, `MODIFY`, and `GET` functions and a standard User should only be able to access the `GET` function.

In the User Interface (UI), the standard user would only be able to see the `GET` function but could in a vulnerable system still send requests to the functions behind the UI. Normally, only an admin user could access the `DELETE` function, but due to the privilege escalation vulnerability, a standard user might also be able to. The HTTP request to delete an account might be as below:

```
POST www.example.com/account/delete
Cookie: SessionID=ADMIN_SESSION

USERID=12345
```

However, due to no check being done on the actual request regarding the session ID (instead of relying only on the UI to stop a non-admin user), a standard user can send a similar request:

```
POST www.example.com/account/delete
Cookie: SessionID=STANDARD_USER_SESSION

USERID=12345
```

When sending the same request, an application that does not check the user session would in this case be vulnerable to privilege escalation, whereas one that checks the

user session and denies the request would not be. Similarly, an application could be vulnerable if the check is incorrectly executed.

2.2.9 Authentication Bypass

Authentication bypass is when an attacker may skip, avoid or abuse an authentication scheme [22]. This can be used in order to get access to resources or request an application to execute code that it normally is not supposed to.

One example is the client-side login in Listing 3 which redirects the user if they provide the correct password. The password can either be found by inspecting the JavaScript file or simply bypassed by moving directly to the path `/admin-interface.html`.

```
function login(password) {
    if (password == "secret-p4ssword") {
        window.location = "admin-interface.html";
    } else {
        alert("Bad password");
    }
}
```

Listing 3: Example of an insecure client-side login.

2.2.10 Indicators of Compromise

Indicators of Compromise (IOCs) are forensic traces showing that the service may already have been compromised by an attack or malware [26]. They are used to detect security incidents and can help understand what events led to the service's compromise.

These traces can be the existence of files, that a server responds in a certain way, or that the content of a web page has been changed. Examples of such in web pages are script inclusions that contain malicious code, web shells being publicly accessible, and references to websites containing malware.

2.2.11 Bad Practice

This category contains wrongful practices in a very general sense. An example of bad practice would be exposing a login portal that is supposed to only be accessible within an internal network and thus opening up attack vectors for malicious actors. Another general type of bad practice would be information leakage, which encompasses whenever some resources are exposed to unintended parties [27]. These resources can be anything from passwords to business documents. A leakage may occur because of a lack of authorization mechanisms (or broken ones), bugs in a program, or a user's fault. The client-side login in Section 2.2.9 exemplified how anyone could access an admin interface without a password.

Another example of information leakage on websites is public directory listings showing the server configuration, secret keys, or other private files. A third example

is a website that gets data depending on a path parameter in the URL, which can be switched out to receive another record that the user should not have access to.

2.3 Amazon Web Services

Amazon Web Services (AWS) are cloud computing services that enable flexibility and scalability for computing, storage, and other functions [28]. By computing through AWS, it is possible to perform tasks that would otherwise be to time or computationally expensive. In this thesis, several services were used to facilitate the huge amount of data processed and analyzed. The services used are presented to give the reader a better understanding when the software architecture is presented later.

AWS was used as the cloud computing resource in this thesis due to a credits grant received by the department. Whilst other cloud computing services could have been used instead of AWS, those were not explored in this thesis.

2.3.1 Lambda

Lambda is a service that runs code without the need to configure a server [29]. A Lambda function has one entry point but can be triggered by multiple sources. The function then executes the uploaded code for a maximum of 15 minutes before timing out.

Each function can be customized regarding its performance, where each instance can have between 128 and 10240 MB of memory allocated to them, with other resources scaling proportionally to the configured memory amount. By customizing the size of the Lambda functions the performance cost can be optimized to perform the calculations faster but with a heftier cost or slower with a lower cost.

Lambda functions were a vital part of our solution as they allowed us to scale the processing of the Common Crawl data by running more Lambda instances simultaneously thus cutting the time required for analyzing the data drastically.

2.3.2 Simple Queue Service

Simple Queue Service (SQS) is, as the name mentions, a queue system that enables message processing in a serverless way [30]. A queue can hold an unlimited amount of messages and must be polled in order to release messages. There are two different kinds of queues to choose between; one alternative is a first-in-first-out ordered queue, and the other is a standard queue without any ordering guarantees. This allows us to queue batches of messages containing all the segments from Common Crawl.

By utilizing the queue, a large amount of Lambda instances could fetch messages containing the necessary information for each segment and start processing them. Once an instance finishes its computation for that batch of messages, it could ask the SQS for further messages until all the messages have been processed.

In order to facilitate any errors occurring during the processing and enable retries in case of failures, the operations performed by the lambda instances need to be idempotent. Hence, checks need to be created to ensure a message from the SQS to the Lambda instances is only processed once.

2.3.3 Simple Storage Service

Simple Storage Service (S3) is an object storage with scalability [31]. One object storage is called a bucket in S3. Additionally, each bucket can be configured to reside in different geographical regions, whether they are publicly available, etc. There are different types of storage classes depending on the use cases to optimize efficiency for long-term storage versus instant access.

The data from Common Crawl is stored in one of these S3 buckets. The data buckets have certain restrictions to them regarding the number of requests that can be handled, and thus redundancy needs to be built in to handle this. AWS recommends the following:

- Use a retry mechanism in the application making requests.
- Configure your application to increase request rates gradually.
- Distribute objects across multiple prefixes.
- Monitor the number of 5xx error responses.

Since the buckets are handled by Common Crawl, the only options available to us are to use retry mechanisms and utilize as few requests as possible.

2.3.4 DocumentDB

DocumentDB is a NoSQL database that consists of JSON documents [32]. The service can consist of clusters to provide high availability and backups with variable sizes to manage different amounts of workloads. Since the results expected from this thesis would not require any type of relational data, a NoSQL database with good performance of reads and writes is optimal.

DocumentDB fit this well and was therefore the database service utilized in our thesis. The base cluster size was sufficient for the scanning part of the thesis but needed to be scaled up to a larger and computationally faster instance when data analysis was performed.

2.3.5 Elastic Compute Cloud

Another service that AWS provides is Elastic Compute Cloud (EC2) instances. These are simple and scalable instances that are built to be high-availability computing resources [33]. While the Lambda instances perform most of the computing power utilized in this thesis, an EC2 instance is still required.

Since the DocumentDB database is run completely in AWS' private cloud, it can't be accessed directly from outside. Therefore, a minimal Elastic Compute Cloud (EC2) instance is used to create an SSH tunnel that can be used to access the database.

2.3.6 Virtual Private Cloud

Amazon creates Virtual Private Clouds (VPC) which are logically isolated virtual networks to act as traditional networks used in on-premise data centers [34]. The above-mentioned services are all run in an Amazon VPC, which means that they do not access the internet but rather just communicate with each other. There are some access points that have to be configured in order for us to be able to interact with the system. These are the EC2 instance that works as an SSH tunnel to reach the DocumentDB instance to view and work on the data and the SQS queue that receives inputs with which Common Crawl segments to run the Lambda code on.

3

Method

To detect vulnerabilities in the static data from Common Crawl, it is necessary to understand how different libraries, plugins, and programs leave fingerprints. By examining how these fingerprints look on web applications and in HTML pages, we can determine how to parse this information from the data and then detect the vulnerable versions and any indicators of compromise.

Whilst it cannot be readily determined that a page is exploitable due to not knowing whether the vulnerable part of that software version is being used, it would still be beneficial for the owner to be aware of the possible risks the page is exposed to. An example would be a library with a vulnerability in one method for a specific version. Whilst a page could be running the vulnerable version, it would not be exploitable until it also uses the method with the vulnerability. It is therefore essential to note the difference between vulnerable and exploitable systems.

3.1 Vulnerability Identification and Fingerprinting

The first step is finding possible ways to identify vulnerabilities in web pages. These will then be categorized, and at least one example of each category will be specified. Vulnerability databases such as the previously mentioned CVE Program [1] will be used to find these possible vulnerabilities. From there, suitable vulnerability indicators can be found, which allow them to be detected.

Before analyzing a large quantity of CVEs and known vulnerabilities, certain elements can help determine the information required. Examples of such elements that will be looked at are meta generator tags, script imports, and common error messages. Using CVEs or other sources, fingerprinting information can be found, and vulnerable websites can be identified.

3.1.1 Meta Generator Tags

Meta tags are HTML tags that contain metadata information regarding programs or plugins run on a website [35]. These tags can point to information that can be used to determine versions of these plugins. The `content` attribute, when used with `name="generator"`, specifies the software which generated the page. An example of a web page running the plugin *WooCommerce* could have a metadata tag as shown in Listing 4.

```
<head>
    <meta name="generator" content="WooCommerce 4.0.1"/>
</head>
```

Listing 4: Example of a meta generator tag for the WooCommerce plugin.

From this metadata, we can easily see that the version of *WooCommerce* is 4.0.1. If there are CVEs for that specific version of *WooCommerce*, it can be determined that that web page might have vulnerabilities.

3.1.2 Web Page Scripts

Like meta tags, a web page can include scripts containing information pointing to specific plugins or programs being run on that web page [36]. Just like for meta generator tags, the versions and names of these plugins are often visible and can be used to determine that the system might be vulnerable, such as in the example in Listing 5.

```
<head>
   <script type='text/javascript'
    ↪  src='.../plugins/.../woocommerce.min.js?ver=4.0.1'
    ↪  id='woocommerce-js'>
   </script>
</head>
```

Listing 5: Example of a script tag for the WooCommerce plugin.

Once again, both the plugin name *WooCommerce* and its version can be parsed, and a potentially vulnerable system can be detected. Notably, the plugin name is available in both the `src` and `id` attributes of the `script` element, while the version is only available in the `src` attribute.

3.1.3 HTTP Headers

HTTP headers are included in the response when requesting a web page. These headers can consist of information such as the server version and authentication protocols [37]. This information can be used to find pages being run on vulnerable servers. An example of how these HTTP headers might look is seen in Listing 6, where the server version can be found quite easily.

```
HTTP/1.1 200 OK
Server: Apache/2.4.1 (Unix)
Vary: X-CDN
Cache-Control: max-age=0
Content-Type: text/html
Date: Sat, 02 Aug 2014 09:52:13 GMT
```

```
Expires: Sat, 02 Aug 2014 09:52:13 GMT
Connection: close
Set-Cookie:UID=...; expires=Sun, 02-Aug-15 09:52:13 GMT; path=/;
↪ domain=example.com;
```

Listing 6: Example of the header data returned for a web page.

3.1.4 Links

Links on a web page may point to malicious content. This is often the case when an attacker already controls the page. They may then update links to point to other pages containing malware or other unwanted things. In other cases, a link may indicate that some specific application is used to deliver the content on the web page. It can then be fingerprinted by finding patterns that an application always uses when creating links. An example of a link generated in HTML is seen in Listing 7.

```
<body>
    <a href="https://example.com/some-link">Some link</a>
</body>
```

Listing 7: Example of how a link can be created in HTML.

3.1.5 Page Titles

Page titles are the information displayed in a browser's title bar or on the tabs [38]. The data is pure text, as in the example in Listing 8, and can easily be matched against other strings. Pages of interest for the thesis that can be found through this information should not be publicly accessible, e.g., company login portals or directory listings. Even though the pages might not be insecure, leaving them public could open up to brute force attacks or denial of service attacks due to the disclosure of the page itself.

```
<head>
    <title>Some title</title>
</head>
```

Listing 8: Example of how a title is defined in HTML.

3.1.6 Keyword Indicators

Keywords can be used to find information of interest that is perhaps different from a vulnerability indicator. For example, OWASP has a list containing a few examples of "juicy information" such as comments in the source code, which could include passwords or debugging information [39]. There are also common search terms that use Google's built-in tools (often called Google dorks) to find vulnerabilities. A list of dorks, sometimes also including exploit code linked to the application the dork

searches for, is maintained in the Google Hacking Database (GHDB)[1]. Using the word list is a fast way to determine keyword vulnerabilities. Furthermore, keyword searches can find content that has been left by previous attackers in the form of indicators of compromise, as described in Section 2.2.10. One such example is shown in Listing 9, where a trace of a web shell is visible in a paragraph.

```
<body>
    <p>Lorem Ipsum</p>
    <p>Malicious Web Shell v13.3.7</p>
</body>
```

Listing 9: Example of a paragraph defined in HTML.

3.1.7 Error Message Information

Continuing from keywords, a subset of those are the ones containing error messages. Best practices for web development are that exact error messages should not be reflected to the end user because they often reveal information that may be abused [40]. An example of this is SQL errors. Those may indicate the possibility of SQL injection attacks. One can, for example, search for common SQL error messages, such as the phrase ”SQL syntax” which is part of the common error message:

> ’You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near [...].’

As SQL injections are one of the application security risks highlighted by OWASP in their Top 10 list [41], this would be one of the most valuable vulnerabilities to look for.

3.1.8 Multiple Criteria

Finding a version, link, or script to verify a vulnerability is usually insufficient. Several vulnerabilities require that multiple criteria are met for the vulnerability to be exploitable. This can, for example, be that a specific version of a software library is in use together with some setting being configured in a particular way. If both criteria are visible in an HTTP response, it is possible to crosscheck and determine if such vulnerabilities are present.

3.2 Parsing Data

The data available from Common Crawl comes in different formats and with varying amounts of information. By examining the various formats and what data can be extracted from each, a suitable data analysis method can be created.

[1] `https://www.exploit-db.com/google-hacking-database`

3.2.1 Parsing Data from WARC Files

Due to the static nature of the data set, only certain parts of the WARC are of interest, and thus the size of the data could be cut down substantially by first extracting the important parts. When analyzing the data received from Common Crawl and typical ways to fingerprint software, plugins, and more, it was determined that the following data could be interesting whilst also being available in the WARC records:

- Server information from HTTP headers
- HTML title
- Meta generator tags
- Script sources
- Links

By extracting only the data of interest, the information size of one segment decreases from approximately 1000MB to 60MB (compressed data size). The process of the data extraction using the AWS services is seen in Figure 3.1.

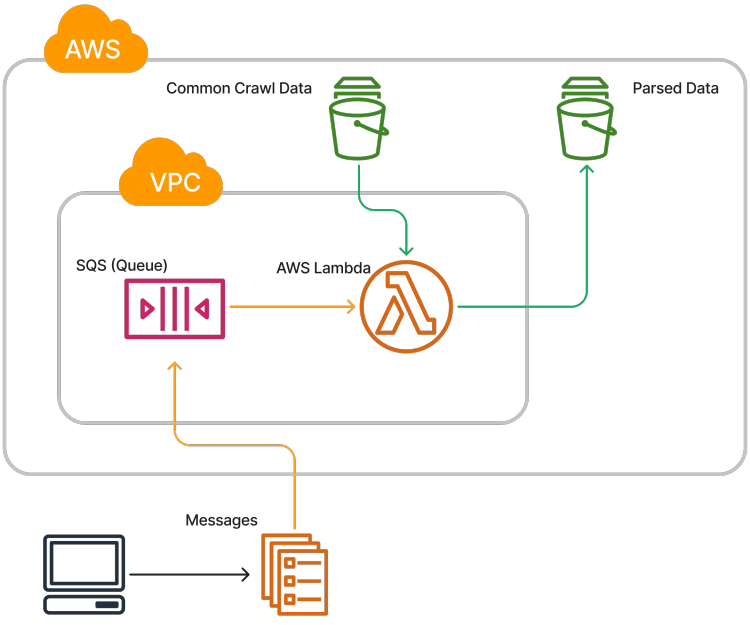

Figure 3.1: AWS process for parsing WARC files from the Common Crawl data.

The process starts with a script generating messages sent to the Simple Queue Service (SQS). These messages contain the S3 storage locations of the different segments parsed for data. The SQS then places these messages in a queue and sends the

messages on to the next available Lambda instance. The Lambda function can be configured to accept both a batch of messages and have a set amount of instances running. These Lambda instances start by getting the segment from the locations sent in the SQS message. This segment is then parsed for the data of interest, which is then saved. The Lambda instances then poll new messages from the SQS and continue this loop until all messages have been handled.

We started with some test runs where the data was saved in a DocumentDB instance. As each segment resulted in roughly 370MB uncompressed (60MB compressed) of data and DocumentDB not supporting any storage compression, one snapshot would result in $370\text{MB} * 88\,000$ segments $= 32,56\text{TB}$ extracted data. This is a large amount of data to save in a database where no database functionality is needed. The saving was therefore changed to write a (~60MB compressed) JSON file to an S3 bucket where storage is cheaper.

3.2.2 Parsing Data from WAT & WET Files

After investigating the data available in a WARC file and what vulnerabilities can be detected, the information was compared to the already extracted information in the WAT and WET formats. Most of the information of interest was already available in processed forms. The scripts, meta tags, links, etc., had all already been extracted into an easily digestible JSON format, with only a few missing data entries from the data of interest, such as the `id` tag for scripts. Utilizing WAT and WET files significantly reduces the time to process the data and storage space. Extracting the information of interest only requires minimal code to access the data in the vulnerability scan, either by navigating the JSON structure in WAT records or by reading the text contents from WET records.

3.2.3 Comparison of Parsing Approaches

Both parsing methods utilizing either WARC or WAT/WET work and should produce the same results. Still, both have positive and negative consequences, as highlighted in Table 3.1.

Table 3.1: Pros and cons of different parsing approaches.

WARC	
Pros	Cons
Tailored data extracted	Computationally expensive to extract
Reduced data size	Expensive cloud storage required
Quicker vulnerability scans	Fixed source data throughout project
	Keywords/SQL errors could not be altered
WAT + WET	
Pros	Cons
No additional storage required	Slower vulnerability scans
Flexible data	Less information available
	Certain data points missing

By extracting the data from the WARC files, the exact data of interest can be found and saved. This would make the vulnerability scans quicker as only the required data would need to be accessed. It would be computationally expensive to extract this data and thus would preferably only be done once. This would lead to a static data set that can not be changed if additional data is required.

The WAT/WET file approach has more or less the opposite consequences, where no additional data storage would be required as the same data can be found in those files. Since the WAT/WET files are much smaller than their WARC counterpart and are better structured, it is relatively easy to extract the data for the scan whilst performing it. Thus, the execution time for the scans does not increase drastically. Considering that no data is extracted ahead of time, the data is more flexible, and new ideas using new data can be implemented without having to parse the files again. However, it was noted that some data, specifically the `id` attribute of script tags, were not saved in the WAT/WET files, leading to a few vulnerabilities being missed when running the comparison. However, this was minor; thus, the advantages of the flexible data and not having to pay for the additional storage were more critical factors in deciding to focus on this approach for the scans.

3.3 Scanning for Vulnerabilities

As different parsing approaches were evaluated, the vulnerability detection methods also had to be altered. When scanning using data parsed from WARC files, the data would be extracted and tailored for the vulnerability scanner. For each different kind of data, such as titles, meta tags, scripts, etc., the appropriate detection method could then automatically be selected to find vulnerabilities. However, the data has not been preprocessed to fit this application when utilizing WAT and WET files. Thus the WAT and WET files are streamed from AWS, and then the interesting data has to be located before the detection methods can be utilized to detect the vulnerabilities. The detection methods are almost identical, whereas the input data is the most significant difference when comparing the approaches.

3.3.1 Scanning from Parsed WARC Data

Once the WARC parser had reduced the data, it could be examined and scanned for some example vulnerabilities. The vulnerabilities were manually selected and inserted into a database. Once again, AWS services were used to speed up the process and examine the data as seen in Figure 3.2.

The scanning utilized many of the same services as in the parsing process. It starts with a script sending S3 links to the parsed data to an SQS queue. This then connects with the Lambda function instances that get the data from S3 and the vulnerabilities that should be scanned for. These vulnerabilities are stored in a DocumentDB instance as different vulnerability types with varying detection methods. The Lambda instances then run the detection methods for the various vulnerabilities and store any detected vulnerability alongside the website data it was seen with, in DocumentDB.

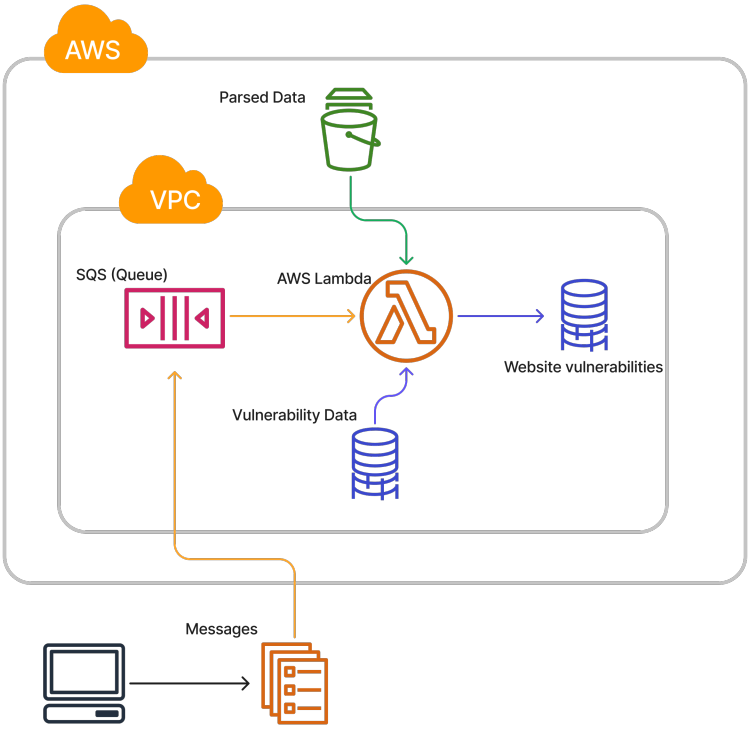

Figure 3.2: AWS process for scanning the parsed data.

3.3.2 Scanning with WAT and WET Data

When scanning for vulnerabilities using WAT and WET files, instead of getting the parsed data from a data source directly, the WAT and WET files are streamed from S3 buckets. Since the data structure matches our requirements, it can be easily digested and used without sacrificing much performance. The data from the WAT files are stored using the JSON format, and thus the important data can easily be extracted and the detection methods can easily process the data. Since the WET file contains plain text, it too can be efficiently used to search for any specific data of interest. Since this text data in the WET file could not be saved from the WARC file without requiring a lot of storage space, keywords and other vulnerabilities searched for would have to be known during the parse and could not be changed without rerunning the parser. However, since the WET files are explored during the scan, the keywords and other vulnerabilities could be updated throughout the project.

The WAT & WET scan follows a similar path as the WARC scanner, where messages containing the S3 links are created and sent to an SQS. The SQS queue sends these messages to the Lambda function instances, which get the WAT and WET files and runs the detection methods for each record in these files. The vulnerabilities scanned for are gathered from the DocumentDB, and any hits on these vulnerabilities are

also saved to the database. The process is illustrated in Figure 3.3.

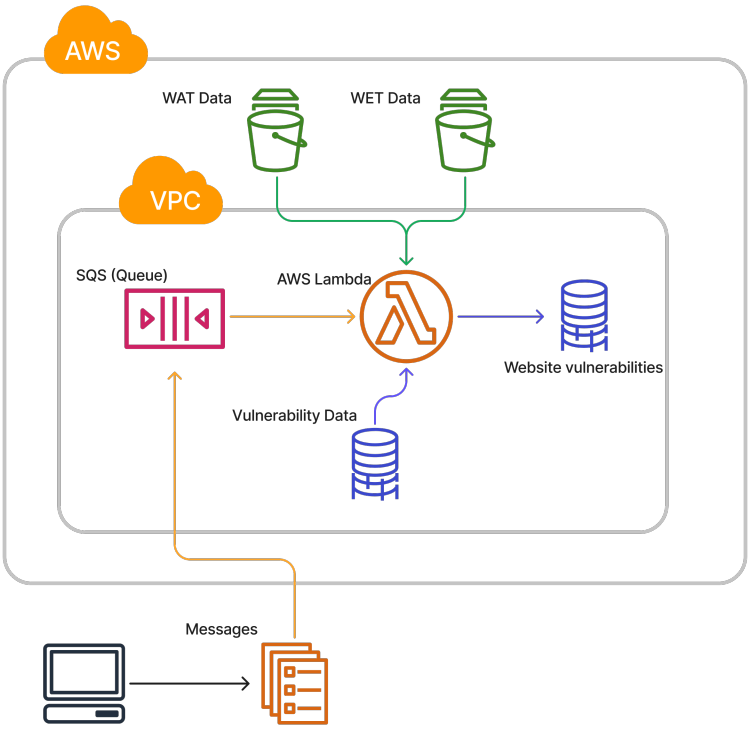

Figure 3.3: AWS process for scanning for vulnerabilities using WAT & WET files.

3.3.3 Comparison of Scanning Approaches

The approaches taken in the different scanners are similar, with the parsing strategy forming how they are performed. The detection methods are also similar, but the input data changes due to the output data from the different parsers. The WET and WAT scanner takes slightly longer since it has to stream two files from an S3 bucket compared to only one for the WARC scanner. However, it allows for keyword and text scans that can help find more vulnerabilities and is not limited to the data searched for during the parse (as the parse is done during the scan).

3.4 Detection Methods

Thanks to the structured data from the crawler, we can use simple and easily maintainable detection methods. For WAT/WET scanning, all detection methods use WAT files to find the vulnerability indicators except for keyword and error message detection, which use WET. The detection methods that use the WAT files

utilize different fields in the WAT records to detect vulnerability indicators. Recall from Section 2.1.4 that the contents of each WAT record are in a JSON format.

3.4.1 Meta Generator Detection

The meta generator tags for each web page can be found in the `HTML-Metadata.Head.Metas` field. Listing 10 highlights an example of the field within the WAT record. This field contains a list of meta tags with the HTML attributes as key-value pairs.

```
{
    "Headers": {...},
    "HTML-Metadata": {
        "Head": {
            "Metas": [
                {
                    "name": "OriginalPublicationDate",
                    "content": "2004/01/22 00:48:49"
                }
            ],
            "Scripts": {...},
            "Title": {...}
        },
        "Links": {...}
    }
}
```

Listing 10: Example of a meta tag from the BBC article shown in Figure 2.2.

By taking this list and checking all entries where `name=generator` and content contains the meta tag of interest. If this meta tag is found, the version will be parsed from the content string if possible. If the version matches the vulnerability criteria for that specific meta tag, it will be noted as a vulnerability in the database.

3.4.2 Script Detection

When checking for vulnerability indicators in script fields, the interesting data is in the URLs. This field usually holds the path to the script file being run. From this path (which can be either local or from another web page), it is generally possible to determine what library/framework is included and its version.

```
{
    "Headers": {...},
    "HTML-Metadata": {
        "Head": {
            "Metas": {...},
            "Scripts": [
                {
```

```
                    "path": "SCRIPT@/src",
                    "type": "text/javascript",
                    "url": "http://www.bbc.co.uk/radio/aod/radioplayer.js"
                }
            ],
            "Title": {...}
        },
        "Links": {...}
    }
}
```

Listing 11: Example of script data from the BBC article shown in Figure 2.2.

The name of the script and the version can then be compared to the vulnerabilities being scanned for, and any matches are then added to the database for this record.

3.4.3 Header Detection

The headers returned and saved in the WAT files follow the `"key": "value"` format. As it is known how different information is written in the headers, they can be scanned for declarations of vulnerable versions of the services. In the example shown in Listing 12, it can be seen that the server is running Apache version 2.4.29. This version is vulnerable to several attacks and can thus be detected[2].

```
{
    "Headers": {
        "Server": "Apache/2.4.29 (Ubuntu)",
        "Date": "Sat, 02 Aug 2014 09:52:13 GMT",
        ...
    },
    "HTML-Metadata": {
        "Head": {...},
        "Links": {...}
    }
}
```

Listing 12: Example of HTTP header data from a vulnerable website.

Whilst not all headers include application versions, a large portion of server headers do and can be used to detect if a web page is being run on an outdated and vulnerable web server.

3.4.4 Link Detection

All the links found on a website are stored in an array, and the `url` field can be used to identify malicious links. An example of a non-malicious link from the BBC

[2] `https://httpd.apache.org/security/vulnerabilities_24.html`

example can be seen in Listing 13.

```
{
    "Headers": {...},
    "HTML-Metadata": {
        "Head": {...},
        "Links": [
            {
                "path": "A@/href",
                "title": "Home of BBC Sport on the Internet",
                "url": "http://news.bbc.co.uk/sport1/hi/default.stm"
            }
        ]
    }
}
```

Listing 13: Example of link data from the BBC article shown in Figure 2.2.

Similar to script vulnerability detection, links can be used to identify malicious or compromised websites, and by scanning these and comparing them to known malicious websites, they can be detected.

3.4.5 Title Detection

Certain information that should not be disclosed can be found by searching the titles for snippets. By searching for login portals or "index of", pages can be found that in themselves are not vulnerabilities but can contain information that is sensitive or that can lead to vulnerabilities such as path traversal. An example of a title in a WAT record can be seen in Listing 14.

```
{
    "Headers": {...},
    "HTML-Metadata": {
        "Head": {
            "Metas": {...},
            "Scripts": {...},
            "Title": "BBC NEWS | Africa | Namibia braces for Nujoma exit"
        },
        "Links": {...}
    }
}
```

Listing 14: Example of title data from the BBC article shown in Figure 2.2.

These can be detected by having a list of commonly used titles for login portals or file index locations and comparing them with the title in the WAT file for each record. If the title matches any of the stored ones, the record is again added to the database.

Compared to many other detection methods, the websites detected through this method need to be examined much further to determine actual vulnerability. They are more often an indicator of bad practice.

3.4.6 Keyword Detection

As previously mentioned, indicators in the form of keywords and SQL errors are detected from WET files. The format was presented in Section 2.1.5 and contains a plain-text extraction of the website contents.

SQL errors and keywords are detected in the same way and are only separated to distinguish the vulnerability category easily. The detection algorithm searches for any text matching the specified keywords in the content (from line 3 in Listing 2) of each record.

3.4.7 Multiple Criteria Detection

As explained in Section 3.1.8, a crosscheck of multiple criteria is sometimes necessary. This can be done by sequentially detecting each criterion, stopping once one criterion can not be found or once all requirements have been detected. Successfully detecting multiple criteria increases the probability of the vulnerability being present and the web application exploitable.

4

Implementation

In order to find vulnerabilities on a large scale, we evaluated different methods to determine the possibility of performing the scan and what vulnerabilities could be detected. We created a proof of concept (PoC) that takes a WARC file and locally parses it and saves the data. This data is then scanned, and vulnerabilities can be detected. Once it was determined that the data from Common Crawl could yield results, a similar setup was performed by utilizing cloud computing. The different parts of the PoC were transferred to AWS and to ensure that the application could be scaled, test runs were executed on the cloud. The different Common Crawl data types were then analyzed to see if computation times could be reduced by utilizing the minimized WAT and WET files produced by Common Crawl.

4.1 Vulnerability Selection

The vulnerabilities selected were chosen to demonstrate the spread of different indicators that one can find from the data rather than finding as many indicators as possible. By choosing a variety of different vulnerability types (RCE, SQLi, CSRF, etc.), that can be detected by a multitude of different detection and fingerprinting methods, a more complete answer to the question of whether or not it is possible to detect these kinds of vulnerabilities on a very large scale (**RQ2**) can be produced. The following criteria were used to find vulnerabilities:

- Vulnerability type
- Detection method
- Time of disclosure

By using the above criteria the vulnerabilities presented in Table 4.1 were selected to be scanned for in the Common Crawl data. The IDs in the table are internal IDs used to map vulnerabilities and will be the same throughout the thesis when referencing specific vulnerabilities.

Table 4.1: The different vulnerabilities selected for the large-scale scan, with information such as affected versions, search strings, and more, in Appendix C.

ID	Name	Detection Method	CVE
3	Duraspace Dspace - Privilege escalation	Metatag	CVE-2021-41189
4	Zblogcn Z-BlogPHP Open redirect	Metatag	CVE-2020-18268
6	Google Dorks - Netgate pfSense Plus - Login	Title	-
7	AngularJS - Prototype pollution	Script	CVE-2019-10768
8	General SQL errors	SQL	-
9	MSSQL errors	SQL	-
10	PostgreSQL errors	SQL	-
11	KILL_THE_NET indicator of compromise	Keyword	-
12	WordPress Core - Path traversal	Metatag	CVE-2008-4769
14	Generic File Listing - index of	Title	-
17	WordPress Houzez Plugin	Script	CVE-2023-26009
21	WordPress Corsa Theme	Script	CVE-2023-23970
22	WordPress Enfold Theme	Script	CVE-2021-24719
23	Violetlovelines IOC	Script	-
24	XSS JQuery UI	Script	CVE-2022-31160
25	Apache HTTP Server Path Traversal/RCE	Header	CVE-2021-41773
26	HTTP Request Smuggling in Apache HTTP Server	Header	CVE-2022-22720
27	Google Dorks - Parallels User Portal	Title	-
28	Vue JS - XSS	Script	CVE-2018-6341
29	Embedthis GoAhead 2.5.0 - arbitrary HTTP Host header	Header	CVE-2019-16645
30	WordPress LearnPress Plugin	Script	CVE-2022-45808
31	Fancy Product Designer plugin	Script	CVE-2021-4096
32	ASP.NET Security Feature Bypass Vulnerability (Asp.net MVC)	Header	CVE-2018-8171
33	ASP.NET Security Feature Bypass Vulnerability (Asp.net Core)	Header	CVE-2018-8171
35	vBulletin Remote Command Execution via widget	Metatag	CVE-2020-17496
36	Joomla! - Unauthenticated SQLi	Metatag	CVE-2022-23797
37	MediaWiki - MassEditRegex allows CSRF	Metatag	CVE-2021-46147
38	Ghost - Allows all authenticated users to view admin API keys	Metatag	CVE-2021-39192
39	React DOM - Lack of escaping could lead to XSS	Script	CVE-2018-6341
40	RCE Error text	Keyword	-
41	PHP Shell/Backdoor - FierzaXploit	Keyword	-
42	PHP Shell/Backdoor - Symlink Sa	Keyword	-
43	PHP Shell/Backdoor - Sole Sad	Keyword	-
44	PHP Shell/Backdoor - TheAlmightyZeus	Keyword	-
45	PHP Shell/Backdoor - RBBD	Keyword	-
46	File zipper	Keyword	-
47	PHP Info leakage	Keyword	-
48	WordPress Core - Authenticated Code Execution	Metatag	CVE-2019-8942
49	VMWARE VSPHERE login portal	Script	-
50	AppleJeus Cryptocurrency Malware	Link	-
51	Leafmailer mail client	Script	-
53	WordPress Enfold Theme - Multi Criteria	MultiCriteria	CVE-2021-24719

For each detection method, a more in-depth description of the vulnerabilities selected will be described.

4.1.1 Meta-generator Tags

Several different generator tags were examined in this thesis. Due to a lot of web applications moving away from utilizing the generator tags in the metadata mostly WordPress and related plugins could be examined using this detection method. WordPress is nonetheless a commonly used system with many different plugins being used on web pages across the world. There are however a couple of other systems that leave meta-generator tags, and these combined with the WordPress ones provide a good spread of different vulnerability types such as SQL injection and privilege escalation as can be seen in Table 4.2. An example of this would be vBulletin which

is a forum service that for versions greater than `5.5.4` but less than `5.6.2` contains a vulnerability that allows remote code execution (RCE) and path traversal (PT) [42].

Table 4.2: Meta-generator tags scanned for and their vulnerable versions and vulnerability types.

ID	Name	Affected Version	Vulnerability Type
3	Duraspace Dspace	= 7.0	Privilege Escalation
4	Zblogcn Z-BlogPHP	<= 1.5.2	Information Leakage
12	WordPress Core	<= 2.3.3 = 2.5	Path Traversal
16	WordPress LearnPress Plugin	<= 4.1.7.3.2	SQL Injection
17	WordPress Houzez Plugin	<= 2.6.3	Privilege Escalation
35	vBulletin	<= 5.6.2 >= 5.5.4	Remote Code Execution
36	Joomla!	<= 3.10.6 >= 3.0.0 <= 4.1.0 >= 4.0.0	SQL Injection
37	MediaWiki	< 1.35.5 <= 1.36.2 >= 1.36.0 = 1.37.0	Cross-Site Request Forgery
38	Ghost	<= 4.9.4 >= 4.0.0	Privilege Escalation
48	WordPress Core	<= 4.9.8 >= 3.7 = 5.0.0	Remote Code Execution

By scanning the meta-generator tags as presented in Section 3.1.1, vulnerable websites can be detected.

4.1.2 Web Page Scripts

The vulnerabilities being searched for in the `script` tags can be seen in Table 4.3, which contains a mix of vulnerable libraries with the version in question as well as some indicators of compromise where the script sources are external web pages distributing malicious scripts.

4.1.3 Links

As mentioned in Section 3.1.4, there can be links pointing to malicious destinations and the web page can thus be assumed to be either malicious or compromised. Two examples are included in Table 4.4. The first one (AppleJeus) searches for websites known to contain malware and the second contains links from a vulnerable plugin.

4.1.4 HTTP Headers

Examples of vulnerable server versions and other headers indicating vulnerability can be found in Table 4.5 below. Both Apache and ASP.NET are two very commonly used servers with a variety of different vulnerabilities associated with them. For this thesis, a few were selected, but further ones could be found through similar means.

Table 4.3: Script information scanned for and their vulnerable versions and vulnerability types.

ID	Name	Affected Version	Vulnerability Type
7	AngularJS	<= 1.7.8 >= 1.4.1	Prototype Pollution
21	WordPress Corsa Theme	<= 1.5	Arbitrary File Upload
22	WordPress Enfold Theme	<= 4.8.1	Cross-Site Scripting
23	Violetlovelines IOC	Not Applicable	Indicator of Compromise
24	jQuery UI	< 1.13.2	Cross-Site Scripting
28	VueJS	< 2.5.17	Cross-Site Scripting
30	WordPress LearnPress plugin	<= 4.1.6.3.2	SQL Injection
31	Fancy Product Designer plugin	<= 4.7.5	Cross-Site Scripting
39	ReactDom	<= 16.1.1 >= 16.1.0 = 16.2.0 <= 16.3.2 >= 16.3.0 <= 16.4.1 >= 16.4.0	Cross-Site Scripting
49	VMWARE VSPHERE	Not Applicable	Login Portal
51	Leafmailer mail client	Not Applicable	Indicator of Compromise

Table 4.4: Links scanned for in HTML.

ID	Name	Vulnerability Type
50	AppleJeus Cryptocurrency Malware	Malware
52	WordPress Avia Page Builder	Cross-Site Scripting

4.1.5 Titles

The title vulnerabilities scanned for in this thesis are not necessarily active vulnerabilities but rather serve as indicators of bad practice or disclosure of information that could be avoided, as per Section 3.1.5. Two login portals and ”index of” pages were searched for, as seen in Table 4.6.

4.1.6 Keywords

In Table 4.7 a mix of different snippets are displayed that can be found through keywords. They either indicate that a website is compromised or that it is vulnerable.

Many of the snippets searched for were chosen as they are commonly used by actors in their malicious code such as the PHP shells or backdoors. So while a non-compromised website might have these snippets, it is more likely that a malicious actor has already comprised the web application.

4.1.7 SQL Error Information

By collecting a set of different, common error messages for specific and general SQL servers (as shown in Table 4.8) and searching for these, websites running a database on the server can be found. As mentioned in Section 3.1.7, whilst these errors are easy to find, they do not confirm vulnerability on the system they were found but

Table 4.5: HTTP headers scanned for and their vulnerable versions and vulnerability types.

ID	Name	Affected Version	Vulnerability Type
25	Apache HTTP Server	>= 2.4.49 <= 2.4.50	Path Traversal/Remote Code Execution
26	Apache HTTP Server	>= 2.4.0 <= 2.4.52	HTTP Request Smuggling
29	Embedthis GoAhead	= 2.5.0	Code injection
32	ASP.NET MVC	= 5.2	Security Bypass
33	ASP.NET Core	= 1.0 || = 1.1 || = 2.0	Security Bypass

Table 4.6: HTML titles indicating vulnerabilities/disclosure of unnecessary information.

ID	Name	Vulnerability Type
6	Netgate pfSense Plus	Login Portal
14	index of	Information Leakage
27	Parallels User Portal	Login Portal

similarly to other vulnerabilities detected in this thesis, serve as good indicators.

For this thesis, a few generic SQL errors such as “You have an error in your SQL syntax”, and more specific ones for MSSQL and PostgreSQL, were chosen. There are a lot more that could be scanned for, but as these only work as indicators, the ones chosen will work as examples of what could be used.

4.1.8 Multiple Criteria Combinations

In order to find vulnerabilities that require multiple criteria to be fulfilled to be exploitable, more advanced methods to detect them were required. These indicators

Table 4.7: Keywords in the HTML text indicating vulnerabilities/disclosure of unnecessary information.

ID	Name	Vulnerability Type
11	KILL_THE_NET	Indicator of Compromise
40	RCE error text	Remote Code Execution
41	PHP Shell/Backdoor - FierzaXploit	Indicator of Compromise
42	PHP Shell/Backdoor - Symlink Sa	Indicator of Compromise
43	PHP Shell/Backdoor - Sole Sad	Indicator of Compromise
44	PHP Shell/Backdoor - TheAlmightyZeus	Indicator of Compromise
45	PHP Shell/Backdoor - RBBD	Indicator of Compromise
46	File zipper	Information Leakage
47	PHP Info leakage	Information Leakage

Table 4.8: SQL texts scanned for indicating vulnerabilities relating to SQL attacks.

ID	Name	Vulnerability Type
8	General SQL errors	SQL Injection
9	MSSQL errors	SQL Injection
10	PostgreSQL errors	SQL Injection

of vulnerability would normally require multiple individual steps to verify that the vulnerability exists and is exploitable, but by combining these, it can be done automatically.

We searched for one combination, as seen in Table 4.9. The script indicator that was used as an example was the WordPress Enfold Theme (vulnerability ID 22, Table 4.1). The vulnerable version of Enfold can be found by first checking for the standard script information. The second criterion is detected through a link indicator, the Avia Page Builder (vulnerability ID 52). This builder can be detected by checking if links on the page contain the `avia-element-paging` parameter. If both of these parts are detected the chance of false positives decreases and the likelihood of the cross-site scripting vulnerability being exploitable increases greatly.

Table 4.9: Multiple criteria vulnerabilities searched for.

ID	Name	Criteria 1	Criteria 2	Vulnerability Type
53	Enfold Theme	Enfold Theme Version (ID 22)	Avia Page Builder (ID 52)	Cross-Site Scripting

4.2 Proof of Concept

In order to verify that the data could be used to detect vulnerabilities a proof of concept had to be created. This proof of concept contained a parser for the Common Crawl data and a vulnerability scanner mentioned in Section 3.3 and Section 3.2. The parser was used to parse the important data from a single WARC file containing approximately 36 000 web pages. The data parsed from these web pages could then be run through the scanner with a list of example vulnerabilities and store the results in a database as seen in Figure 4.1. We used ~15 vulnerabilities in the scan, mainly to test the validity of the different detection methods.

This proof of concept, due to the limited data size, could be run locally and used to verify that it was indeed possible to find vulnerabilities using the data available from Common Crawl. The proof of concept was then utilized to set up a similar pipeline using cloud computing to perform the resource-heavy tasks of parsing and scanning an entire snapshot.

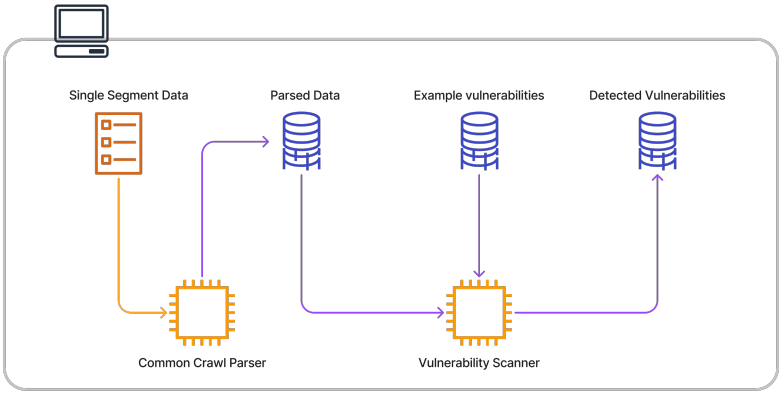

Figure 4.1: Proof of concept run on local computer.

4.3 Utilizing Cloud Computing

Due to the large data sets provided by the Common Crawl project even for a single snapshot, performing the data analysis would not be feasible on a standard computer. In order to combat this, Amazon Web Services (AWS) was used to speed up the computing tasks in this thesis. Several different services (which were presented in Section 2.3) provided by AWS were used in this project:

- Lambda - used as the computing resource to speed up the computing time when scanning for vulnerabilities.
- Simple Queue Service - the queue handling the segments of the Common Crawl data and the running of the Lambda instances.
- Amazon S3 - the storage service for the Common Crawl data.
- DocumentDB - the No-SQL database used to store the results from the scan as well as hold the vulnerability information.
- Amazon EC2 - the computing instance used to access the database data in the AWS cloud from outside the VPC.

By combining these services in the computing-heavy areas of this thesis, handling large data sets became less of a challenge.

4.4 Changes from Proof of Concept

After running the proof of concept on a small data set and validating the results obtained, it became evident that certain parts were not working as expected or not working efficiently enough.

When validating the data and the results from it, we noted that there were some

issues that would need fixing:

1. Parsing WARC data files taking too long.
2. Having to specify certain vulnerabilities before the parsing instead of before the scan.
3. Certain vulnerabilities not being detected as expected.
4. Idempotent programming of Lambda functions.
5. Retrying on errors.
6. Statistics and better logging.
7. Version fingerprinting not working as expected in edge cases.
8. Efficiency of Lambda functions.

These issues could be fixed in a variety of different ways, and our choices and reasoning is presented next.

4.4.1 Using WAT and WET versus WARC File Format

By testing the efficiency of using WAT & WET files compared to WARC, major efficiency improvements could be made as the program would no longer need to perform the parsing step done by the parser.

The time measurements of 10 segments are shown in Table 4.10 and it can be seen that processing WAT and WET is notably faster for a single segment on average (215% speed increase). However, the WARC scan did yield a few more vulnerability hits, but the trade-off seemed worth it.

Table 4.10: Time comparison of Lambda functions. All functions were configured to have 1536MB of memory.

	WARC			WAT & WET
	Parse (s)	Scan (s)	Combined (s)	Combined (s)
Segment 1	220	142	362	175
Segment 2	206	128	334	175
Segment 3	215	140	355	176
Segment 4	194	145	339	176
Segment 5	219	145	364	146
Segment 6	218	139	357	149
Segment 7	198	129	327	151
Segment 8	200	134	334	155
Segment 9	219	131	350	151
Segment 10	198	131	329	146
Average (s)	209	136	345	160
Cost (GB-seconds)	321	210	530	246
Vulnerabilities found			51877	51546

One example of a vulnerability indicator that was identified in WARC but not in WAT & WET was the following:

```
<script type="text/javascript" src="https://ajax.googleapis.com/ajax⌋
↪ /libs/jqueryui/1.12.1/jquery-ui.min.js?ver=1.12.1"
    id="jquery-ui-core-js"/>
```

This was due to the texts that were searched for being `jquery-ui-core` and `jquery/ui`. One of those was found in the *id* attribute within the script inclusion, which is not included in the WAT or WET extracts:

```
{
    "type": "text/javascript",
    "url": "https://ajax.googleapis.com/ajax/libs/jqueryui/1.12.1/jq⌋
↪ uery-ui.min.js?ver=1.12.1",
    "path": "SCRIPT@/src"
}
```

As can be seen, it would be possible to identify the string `/jqueryui/` or `jquery-ui.min` and as such also detect it in the WAT format. When the indicators were set up, we had not previously seen this identifier, and therefore it was not included in the search. The same issue exists in general for any vulnerability indicators; it is infeasible to find all possible texts that would identify a vulnerability and the search time would increase substantially with more identifiers.

Using WAT & WET files would also solve the issue of having to know what keywords to parse before the scan could take place and thus by switching the file format issue 1 and 2 could be solved.

4.4.2 Improving vulnerability matching

After having performed the scan for the vulnerabilities mentioned above, we found that some of them generated zero results. When checking this, it was determined that some of the vulnerable libraries/plugins did not generate a fingerprint that could be detected from the data available or that a better more certain fingerprint existed and could be used instead.

By modifying the fingerprint being searched for, more accurate results could be obtained in the final scan remedying issue 3.

4.4.3 Programmatic improvements

Some of the issues noted during the first trial runs were that errors generated could lead to a lot of duplicate results due to the automatic retrying by the Lambda instances as well as the SQS message being picked up by multiple simultaneous Lambda instances.

By improving logging, creating our own retry function for problematic calls, and making the code idempotent a lot of the issues were solved. When correctly logging and retrying failed calls internally, we could avoid saving part of the data and

duplicating it when it was rerun. We also managed to solve the issues caused by the SQS where a message in the queue could be picked up by multiple Lambda instances by ensuring that we checked whether that message had already been processed. Furthermore, we increased the detection rate of vulnerabilities by improving the version detection algorithm. These improvements solved issues 4, 5, 6, and 7, allowing for more accurate results to be obtained.

4.4.4 Lambda Optimisation

Amazon houses an open-source optimization tool, AWS Lambda Power Tuning, for Lambda instances in the Serverless Application Repository [43]. With the tool, the user can set different memory sizes and thus reduce the costs of the Lambda. Since the computational power of a Lambda function scales with the memory size, the execution time changes as well. The cost of a Lambda function execution is determined by the execution time and memory size and thus, by using this tool, the most time and cost-effective size of the Lambda instances can be chosen. The results from running the tool are presented in Table 4.11.

Table 4.11: Time and cost comparison of different Lambda function settings (averages of 10 runs each).

Size (MB)	Average cost	Average Duration (s)	Total cost
3008	$0.0094564	236.7	$0.094141
2048	$0.0065580	241.1	$0.065670
1536	$0.0058443	286.5	$0.057839
1024	$0.0058895	433.0	$0.058964
512	-	-	-

As expected the execution time decreases with higher memory, but the cost increases. There is however a small dip in cost at 1536 MB compared to 2048 MB and 1024 MB, which indicates that the runtime is fast enough compared to the cost to make it worth it to run with more computational power. The 512 MB had no results as it timed out after 600s. Due to the Lambda instances having a max duration of 900s, this might not be a suitable choice even if the cost were lower, which the other results do not indicate.

So by running this tool the efficiency of the Lambda function and our solution as a whole could be optimized to be both cost-effective and fast, reducing the computational time of the large-scale scan.

4.5 Final Setup

By running the proof of concept, testing it on the cloud, and continuously improving it throughout the thesis, a final setup could be established. The scanner used for the large-scale scan of the whole Common Crawl data set consisted of an expanded and improved version of the scanner as shown in Figure 4.2. This program utilized

the WAT and WET files instead of similar data extracted from the WARC files. For each message sent to the SQS, the corresponding segments WAT and WET files are fetched from the S3 bucket and run through a Lambda function instance. The program then uses the detection methods described in Section 3.4 to detect the vulnerabilities stored in the DocumentDB instance. If any of these vulnerabilities are detected they are saved to a result collection in the database.

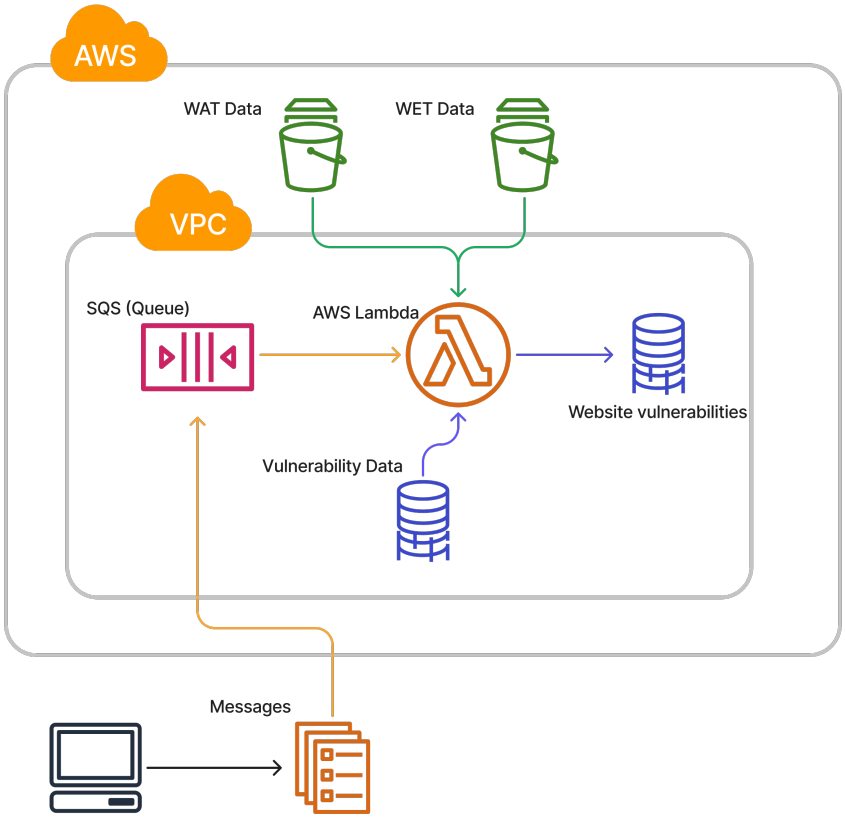

Figure 4.2: AWS process for scanning for vulnerabilities using WAT & WET files.

5

Dynamic Verification

Simply finding indicators of vulnerabilities on websites is not enough to guarantee the pages are vulnerable. To verify whether the vulnerabilities were exploitable or not we created a dynamic verifier. A vulnerable package or framework may be present on a website, but some vulnerabilities also require settings or specific function calls to be exploitable. The idea is that the verifier should be able to detect this non-intrusively.

After the small-scale scan, a few vulnerabilities were tested and verified manually, and after the large-scale scan, selected vulnerabilities and domains were verified automatically with the verifier by utilizing Python scripts. The process is visualized in Figure 5.1 and starts by verifying that the vulnerability can still be detected on the web application. This includes checking if any of the indicators from the crawl is still present, if a vulnerable version has been updated, and is therefore now safe, or if that version still includes the vulnerability. If indicators were found for a vulnerability, further checks were made to verify whether it would be exploitable.

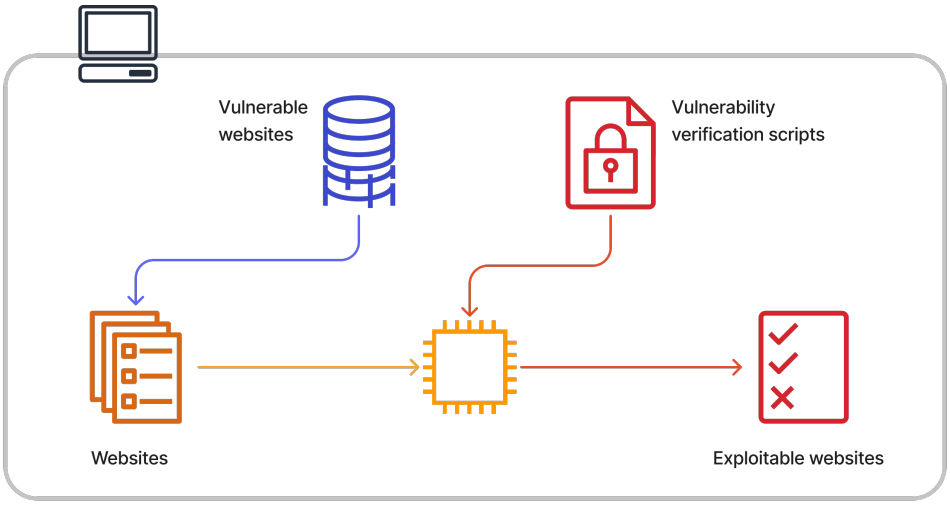

Figure 5.1: Dynamic verification procedure.

To ensure that no damages were caused, we limited the selection of vulnerabilities to

those that could be verified in a non-harmful manner. This includes creating pop-up alerts using XSS, attempting to echo messages during RCE, and only accessing files that do not contain any sensitive information during path traversals, such as public open-source text files.

Apart from the verifications being non-harmful, we selected vulnerabilities that can be verified with a reasonably low attack complexity and which had a spread in both exploit types and the vulnerability source. The ones that were chosen for more rigorous verification are:

- WordPress Enfold theme - Cross-Site Scripting
- WordPress LearnPress plugin - Path Traversal
- vBulletin - Remote Code Execution
- jQuery UI - Cross-Site Scripting
- Apache Web Servers - Remote Code Execution/Path Traversal
- Indicators of Compromise

5.1 Case Study of Detected Vulnerabilities

Validating the vulnerabilities is a tedious process. Making the validation generic is even more so. The case study aims to describe how one can verify that a specific vulnerability is exploitable and showcases a few different vulnerability types.

The success rate of the validations is dependent on several factors. Firstly, the time difference between the snapshot and when the validations were done affects the results. The snapshot used for the scan was from January/February whilst the validations were run at the end of April. Secondly, as the validation scripts were created and tested by us, there is a possibility that they were not generic enough and therefore fail on some of the scanned web pages that in reality are vulnerable. Thirdly, web pages sometimes utilize protective mechanisms outside the application to defend themselves. These protections include Content Security Policy which counteracts XSS and injection attacks [44], and Web Application Firewalls which analyze and block certain traffic before it reaches the web page [45], amongst others. Lastly, web pages or applications may remove the indicators of the software they are running, meaning that they would not be found during the scan which we performed. Some applications present wrong indicators on purpose or have “stealth patches” in their software, meaning that the same version sometimes contains the vulnerability, and other times it has been patched without updating the version number.

5.1.1 WordPress Enfold Theme - Cross-site Scripting

One vulnerability that was found in the scan is the possibility of reflected cross-site scripting when using an old version of the WordPress Theme Enfold together with

Avia Page Builder[1]. It was verified by using Selenium, checking that an alert pops up as shown in Figure 5.2. The vulnerable websites were found using a multi-criteria detection method, searching for `themes/enfold` inside the script source URLs and `avia-element-paging` inside links on the page.

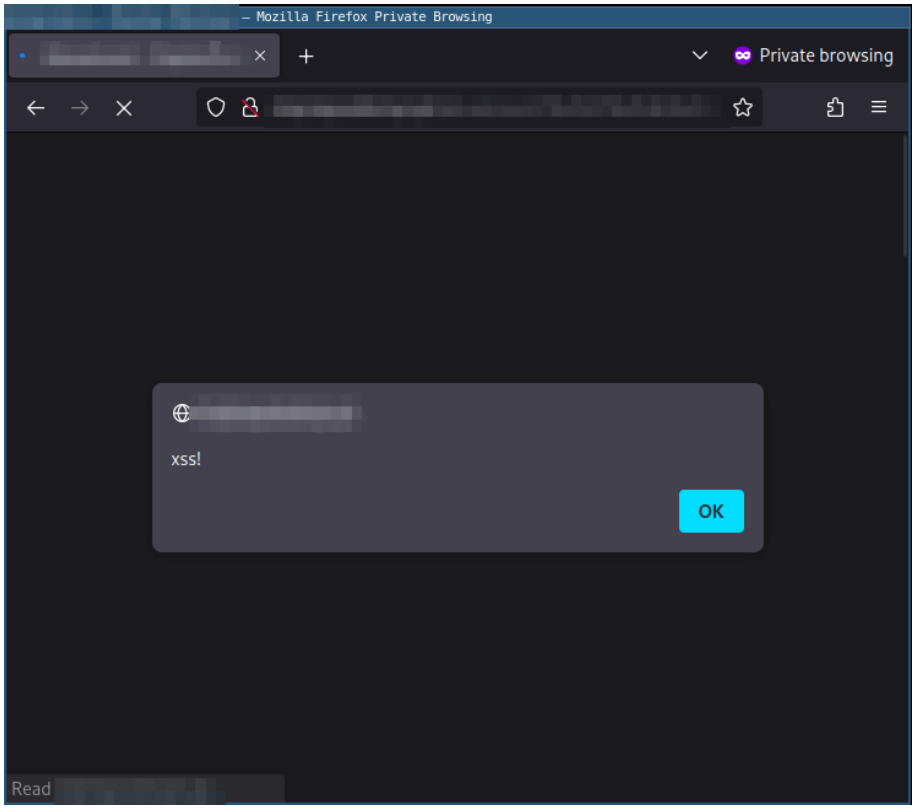

Figure 5.2: Example of a web page vulnerable to an XSS attack, reflecting the text sent to verify the vulnerability.

5.1.2 WordPress LearnPress plugin - Path Traversal/Remote Code Execution

By accessing a specific API endpoint supplied by the LearnPress plugin and sending in a path variable pointing to a file of interest, then the plugin returns this file in the HTTP response. In order to not access any sensitive information we decided to try to access the `license.txt` file. The content of this file could then be checked to ensure that the correct file was returned.

The versions searched for of the plugin also have a remote code execution vulnerability, but we figured that just accessing a normally not accessible file would be enough to verify the exploitability of the plugin.

[1] `https://nvd.nist.gov/vuln/detail/CVE-2021-24719`

5.1.3 vBulletin - Remote Code Execution

Certain versions of the community software package vBulletin are vulnerable to remote code execution by exploiting the code portion in the widgets created by the package. By inserting a sub-widget into another widget code can be executed.

Utilizing a modified version of the proof of concept exploit released with the vulnerability disclosure; we can send in shell commands such as `echo` into the code portion of the widget and verify the vulnerability.

5.1.4 jQuery UI - Cross-site Scripting

Certain conditions need to be fulfilled for the jQuery UI package to be exploitable. First, a checkboxradio button supplied by a specific widget within jQuery UI must be on the web page. Second, the button must have a label not surrounded by a `span` tag and be controlled by some user input. Finally, with both prerequisites in place, the code needs to trigger a refresh on that specific input. Once the widget is refreshed, the HTML code in the label field of the checkboxradio button becomes evaluated as HTML in the clients' browser.

Since this vulnerability has many requirements, the verification checks whether the package is still used. In other words, we do not validate that the application is vulnerable only if it uses a vulnerable package.

5.1.5 Apache Web Servers - Path Traversal and Remote Code Execution

Apache HTTP servers with unpatched versions of 2.4.49 or 2.4.50 are vulnerable to path traversal and remote code execution, depending on the server's configuration. According to the CVE[2] path traversal is possible when directories are explicitly set to be allowed for browsing while RCE requires the path traversal as well as an allowance for CGI scripts in those paths. Nonetheless, payloads available online for the vulnerability allow for quickly automating the verification with non-disruptive shell commands.

5.1.6 Violetlovelines - Indicator of Compromise

The indicator of compromise examined work, as the name suggests, as an indicator that the website with it has been compromised by a malicious user due to other means. There are no specific tests that need to be performed to verify its existence other than checking that the script inclusions linking to any of the Violetloveline domains are still present on the website.

[2] `https://nvd.nist.gov/vuln/detail/CVE-2021-41773`

6

Results

Multiple different results gathered throughout the thesis will be presented in this chapter. Firstly a general description of the results gathered from the complete scan will be presented. Then a review of Top Level Domains (TLDs) will be shown before showcasing vulnerabilities on the top 1 million most visited websites. Finally, the results from the dynamic analysis will be presented. By showing the results from the large-scale scan and the dynamic verification, we can answer both research questions **RQ2** and **RQ3**.

6.1 Hits from Full Scan

After scanning all of the Common Crawl data, certain interesting statics appear. The scan was run on 88 000 Common Crawl data segments utilizing 50 concurrent AWS Lambda functions. As seen in Table 6.1 the computation time for running the scan sequentially would have been close to 244 days. But by utilizing concurrency and the Lambda instances this time could be brought down to 5.5 days, with the possibility of even shorter calculation times by increasing either Lambda instance performance or increasing concurrency.

Table 6.1: Statistics from the first small-scale vulnerability scan.

Data	Value
Segments	88 000
Concurrent Lambda instances	50
Avg. Execution time (s)	240
Total Elapsed time (days)	5.5
Total Execution time (days)	244
Vulnerabilities searched for	42
Avg. No. websites/segment	36 245
Avg. No. vulnerable websites/segment	3 095

Running a scan on 88 000 segments containing approximately 3.2 billion websites gives enough data in order to draw some conclusions. Of these URLs, approximately 273 million of them contained indicators of vulnerability. Without doing any kind of dynamic analysis of the data gathered it can not be determined that these websites are currently vulnerable to the attacks relating to the vulnerabilities detected, but

it can be said that these websites are running versions of plugins, libraries, servers, and more that contain known vulnerabilities.

A vulnerability rate of ~8.5% seems a lot higher than one would first imagine, but it is worth noting that many of the vulnerabilities found occur multiple times on the same domain. The crawler used by Common Crawl scans multiple URLs of the same domain, and thus the same vulnerability will be detected multiple times. These 273 million vulnerable websites match ~3.6 million vulnerable domains, indicating as mentioned, that vulnerabilities found on one certain page of a domain are commonly found on other pages on the same domain. Some domains found have more than 10 000 web pages scanned, of which many contain the same vulnerabilities and might skew the results. Having found ~3.6 million domains out of the 42 million hosts scanned by Common Crawl, indicate a vulnerability rate of ~10.9%. This rate is a bit higher than the corresponding URL rate of ~8.5%.

Whilst 42 vulnerabilities were scanned for, only 39 different vulnerabilities were detected in the data. The counts for each vulnerability on both a URL and domain level can be seen in Table 6.2.

Table 6.2: Vulnerability counts for the full scan.

ID	Vulnerability Name	Webpage count	Domain count
26	HTTP Request Smuggling in Apache HTTP Server	141 224 740	1 262 838
24	XSS JQuery UI	87 486 256	1 603 555
48	WordPress Core - Authenticated Code Execution	23 319 967	535 355
32	ASP.NET Security Feature Bypass Vulnerability (Asp.net MVC)	15 801 296	145 571
37	MediaWiki - MassEditRegex allows CSRF	5 410 974	14 189
22	WordPress Enfold Theme	3 430 339	63 263
14	Generic File Listing - index of	2 836 813	94 308
28	Vue JS - XSS	2 432 374	20 197
7	AngularJS - Prototype pollution	1 393 784	17 166
30	WordPress LearnPress Plugin SQL Injection	408 704	8 619
23	Violetlovelines IOC	316 354	4 936
17	WordPress Houzez Login Register Plugin	282 138	3 504
36	Joomla! - Unauthenticated SQLi	156 013	2 691
25	Apache HTTP Server Path Traversal/RCE	122 028	2 188
8	General SQL errors	108 004	8 290
35	vBulletin Remote Command Execution via widget	77 999	139
12	WordPress Core - Path traversal	75 627	1 625
3	Duraspace Dspace - Privilege escalation	66 006	13
39	React DOM - Lack of escaping could lead to XSS	48 725	1 102
38	Ghost - Allows all authenticated users to view admin API keys	35 730	717
21	WordPress Corsa Theme	34 271	360
31	Fancy Product Designer plugin for WordPress	32 707	1 082
4	Zblogcn Z-BlogPHP Open redirect	25 905	768
47	PHP Info leakage	23 876	3 794
53	WordPress Enfold Theme - Multi Criteria	7 067	1 463
33	ASP.NET Security Feature Bypass Vulnerability (Asp.net Core)	1 935	91
9	MSSQL errors	1 602	275
46	File zipper	1 375	7
10	PostgreSQL errors	971	242

51	Leafmailer mail client	225	9
11	KILL_THE_NET indicator of compromise	139	3
40	RCE Error text	70	29
42	PHP Shell/Backdoor - Symlink Sa	57	35
27	Google Dorks - Parallels User Portal	9	8
43	PHP Shell/Backdoor - Sole Sad	9	5
44	PHP Shell/Backdoor - TheAlmightyZeus	6	6
50	AppleJeus Cryptocurrency Malware	5	5
29	Embedthis GoAhead 2.5.0 - arbitrary HTTP Host header	2	2
41	PHP Shell/Backdoor - FierzaXploit	2	2
6	Google Dorks - Netgate pfSense Plus - Login	0	0
45	PHP Shell/Backdoor - RBBD	0	0
49	VMWARE VSPHERE login portal	0	0

6.2 Top-level Domains

The vulnerabilities found can also be divided up by their respective top-level domains (TLDs). Top-level domains are domains that either indicate the kind of service a site is a part of or what country it is located in. As expected the most common is of course `.com` which makes up approximately 41.5% of all vulnerabilities detected. Other common generic ones include `.org`, `.net` and `.edu` which together make up around ~8.7% of the vulnerabilities with a breakdown of the most common ones displayed in figure Table 6.3.

Table 6.3: Top 10 most common top-level domains with vulnerabilities.

TLD	Count	Percent of vulnerabilities	Percent of CC scan
.com	1 482 764	41,5%	49,9%
.de	211 783	5,9%	7,0%
.org	171 366	4,8%	4,5%
.it	130 212	3,6%	2,9%
.net	112 843	3,2%	3,4%
.fr	105 118	2,9%	2,1%
.ru	100 953	2,8%	2,7%
.nl	94 007	2,6%	2,7%
.pl	83 064	2,3%	1,5%
.cz	55 604	1,6%	1,0%
.es	49 086	1,4%	0,9%

Most of the commonly used TLDs however relate to the country or region that the web page operates from. By cross-checking the vulnerabilities detected with the country codes, a map can be created mapping the TLDs to their respective countries. It is however not enough to just numerically display the count of domains with vulnerabilities for each country since some are more overrepresented than others in the scan performed by Common Crawl. In order to showcase this correctly, it was determined that the results would have to be normalized against the number of

websites scanned by Common Crawl. By dividing the number of vulnerable websites and domains for each top-level domain by their corresponding total scan count a fairer analysis of the top-level domains could be performed.

In Figure 6.1 and Figure 6.2 the normalized top-level domain vulnerabilities can be seen when checking both for individual URLs and for specific domains. The full data for each top-level domain can be seen in Appendix D.

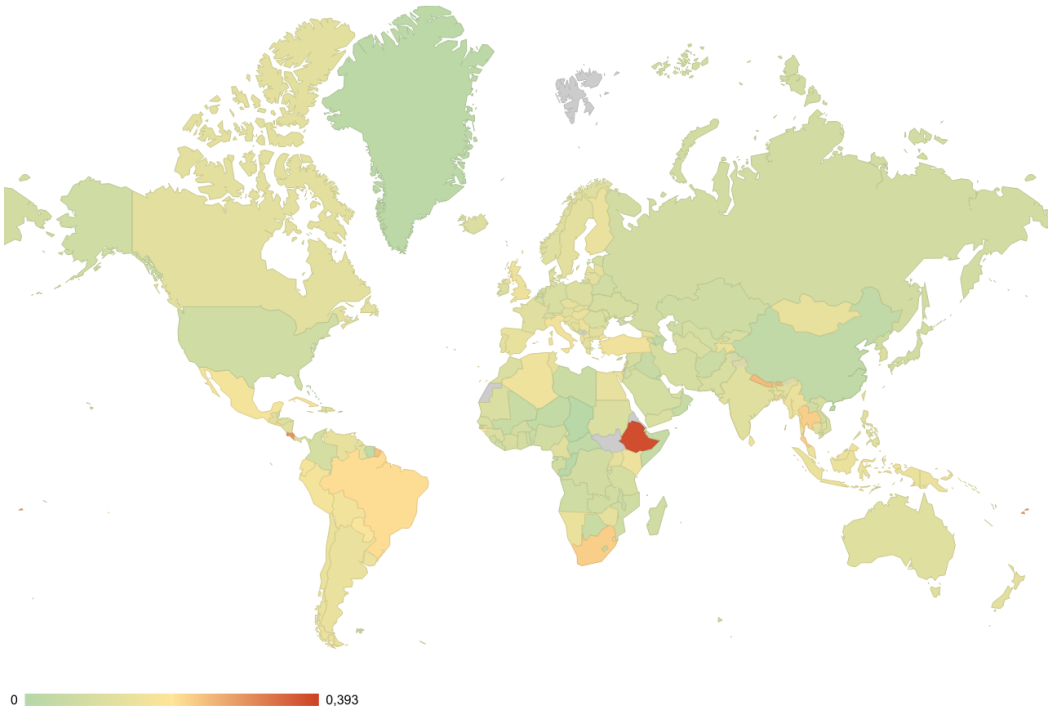

Figure 6.1: Map of vulnerable URLs detected per TLD normalized against the number of URLs scanned by CommonCrawl. An interactive map can be reached here.[1]

[1] https://www.cse.chalmers.se/research/group/security/masters/CCSecurityAnalysis/worldMapURL/

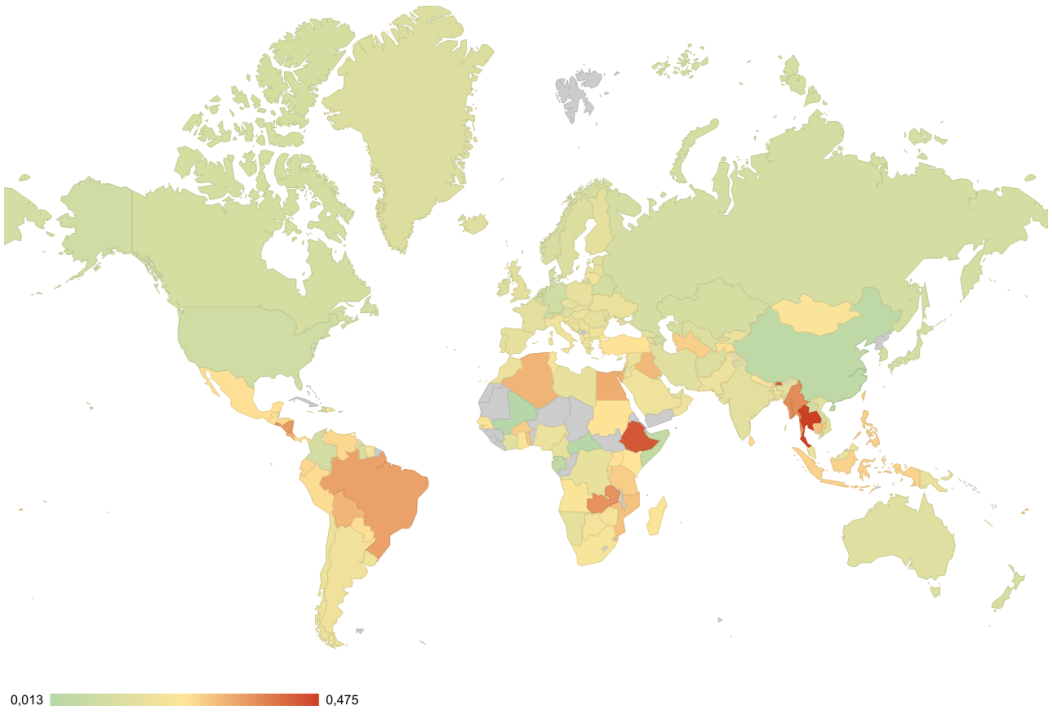

Figure 6.2: Map of vulnerable domains detected per TLD normalized against the number of domains scanned by CommonCrawl. An interactive map can be reached here.[2]

6.3 Vulnerabilities on Top 1M Domains

Other than analyzing the geographical locations of domains with vulnerabilities, it could also be interesting to see how common these vulnerabilities scanned for are on the most visited domains worldwide. To analyze this, the Tranco list was used [46]. The Tranco list consists of the top 1 million most visited domains formed by an aggregation of multiple sources.

By cross-referencing the results from our large-scale scan, domains with any of the vulnerabilities scanned for can be detected. Due to the number of false positives that certain vulnerabilities lean towards, some vulnerabilities will not be included in this data. The vulnerabilities that will be ignored are:

- SQL errors - a lot of false positives due to forum posts and other informational posts about the errors searched for.
- Generic file listings/Index of - a large portion of these are not exploitable, and the data from these listings is often meant to be found.

[2] `https://www.cse.chalmers.se/research/group/security/masters/CCSecurityAnalysis/worldMapDomain/`

- Keyword vulnerabilities - due to a large number of false positives this would not be accurate to include. Most of these served more as possible indicators of compromise and not as specific vulnerabilities in systems and can thus be ignored.

After excluding the above-mentioned vulnerabilities from the data, the vulnerabilities found on the top 1 million domains are shown in Table 6.4. It can be seen that a lot of the vulnerabilities scanned for in this thesis are also present on the most visited websites in the world. It can even be seen that the vulnerabilities are actually even more common as only about 62.8% of the Top 1 million websites were scanned by Common Crawl in the first place.

Table 6.4: Vulnerabilities found on the Tranco Top 1M pages.

Name	Value	Percent
Tranco list size	1M	NA
Scanned by CC	628k	62.8%
Total vulnerable	92 840	9.3%
Top 10 vulnerable	3	30.0%
Top 100 vulnerable	34	34.0%
Top 1000 vulnerable	286	28.6%
Top 10k vulnerable	2 227	22.3%
Top 100k vulnerable	16 125	16.1%
Top 500k vulnerable	55 908	11.2%

Of the vulnerabilities found on the Tranco list, certain vulnerabilities remain more common, quite in line with the results from the overall study. The top three are the Apache Request Smuggling, jQuery-UI, and ASP.NET MVC bypass, with a list of the top ten in Table 6.5.

Table 6.5: Most common vulnerabilities in Tranco Top 1M websites.

ID	Vulnerability	Count
26	HTTP Request Smuggling in Apache HTTP Server	49510
24	XSS JQuery UI	29350
32	ASP.NET Security Feature Bypass Vulnerability (Asp.net MVC)	13040
48	WordPress Core - Authenticated Code Execution	9834
37	MediaWiki - MassEditRegex allows CSRF	2645
28	Vue JS - XSS	2060
7	AngularJS - Prototype pollution	1770
22	WordPress Enfold Theme	1022
47	PHP Info leakage	451
30	WordPress LearnPress Plugin SQL Injection	315
36	Joomla! - Unauthenticated SQLi	121
-	Others	472

6.4 Dynamic Analysis

The dynamic verification results are shown in Table 6.6 and Table 6.7 and as one can see in the rightmost column the results vary a lot between different vulnerabilities. The percentages in the columns *Indicators removed*, *Patched* and *Indicator still existing* is the corresponding number divided by the difference between *Checked* and *Unreachable*. The reason that one row does not have a value in the patched column is that it is an IoC, meaning there is no way to patch it; one can just remove the indicator.

Table 6.6: Dynamic verification results based on URLs.

Vulnerability		URLs					
ID	Name	Checked	Unreachable	Indicators removed	Patched	Indicator still existing	Verified
23	Violetlovelines	1000	468 (46.80%)	404 (75.94%)	-	128 (24.06%)	128 (100.00%)
24	jQuery UI	10000	1507 (15.07%)	373 (4.39%)	2051 (24.15%)	6069 (71.46%)	0 (0.00%)
53	Enfold + Avia	7014	1670 (23.81%)	648 (12.13%)	465 (8.70%)	4231 (79.17%)	1018 (24.06%)

Table 6.7: Dynamic verification results based on domains.

Vulnerability		Domains					
ID	Name	Checked	Unreachable	Indicators removed	Patched	Indicator still existing	Verified
25	Apache 2.4.49	277	10 (3.61%)	7 (2.62%)	5 (1.87%)	255 (95.51%)	0 (0.00%)
25	Apache 2.4.50	189	15 (7.94%)	7 (4.02%)	4 (2.30%)	163 (93.68%)	0 (0.00%)
30	LearnPress	8619	2303 (26.72%)	627 (9.93%)	782 (12.38%)	4907 (77.69%)	877 (17.87%)
35	vBulletin	139	24 (17.27%)	5 (4.35%)	3 (2.61%)	107 (93.04%)	9 (8.41%)

Looking at Figure 6.3, the steps of one verification are visualized. The steps were described in detail in Chapter 5 and can shortly be summarized as:

- Checking if the website is reachable.
- Checking if the website still contains the vulnerability indicators.
- Testing if the website is vulnerable.

6.5 Comparison to existing solutions

In order to give a fair assessment of the value provided by our solution, it is necessary to compare it to similar services. A selection of vulnerabilities used in the scan was selected and searched for using the other services and presented in Table 6.8. In order to search for vulnerabilities on Shodan, the `vuln` keyword with the CVE was used, as well as version-specific free text search phrases. This allowed for multiple versions of programs/libraries to be found simultaneously using the `vuln` keyword, whilst the free text search would find vulnerabilities not mapped to any specific CVE. As PublicWWW does not have a similar search function, versions had to be searched for manually and were sometimes skipped due to having to search for too many versions.

In Table 6.8, it can be seen, that the Apache 2.4.49/50 vulnerability can be detected using all different solutions. Our solution found 2188 vulnerable domains, Shodan

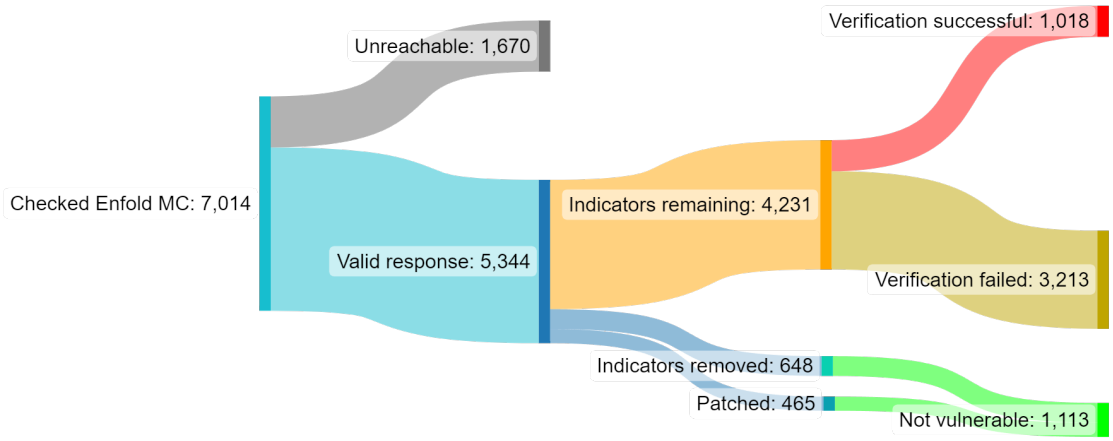

Figure 6.3: Sankey chart of dynamic verification results testing XSS in WordPress Enfold Theme.

Table 6.8: Comparison of results from the different services with data gathered on 2023-05-12. * implies that there are too many versions to feasible search for all.

Name	CVE	CC Scanner	Shodan (vuln)	Shodan (free text)	Public WWW
Apache 2.4.49/50	CVE-2021-42013	2188	5701	5509	1881
AspNet MVC	CVE-2018-8171	145571	0	223534	170870
React DOM	CVE-2018-6341	1102	0	20	580
WordPress Core	CVE-2008-4769	1625	41	*	*
WordPress Enfold + Avia	CVE-2021-24719	1463	0	*	*

found 5701 domains with the `vuln` keyword and 5509 domains with free text search, and PublicWWW only found 1881 domains.

7

Discussion

From the results presented in Chapter 6, most of the vulnerabilities scanned for in the Common Crawl data could be found using the detection methods shown in Chapter 3. As expected, certain vulnerabilities were much more common, as the technology is used more. An example of this would be Apache which is one of the most common types of web servers [47] compared to some WordPress plugins or PHP shells, which are less used. It can be noted that all detection methods found at least one vulnerability, which implies that the detection methods fulfill their job. There is, however, the issue of some detection methods being more prone to false positives due to what data is used to detect the vulnerabilities, further discussed in Section 7.2.

Looking at the top 10 TLDs found in the scan, it can be noted that some types of TLDs perform better at creating more secure web applications. The `.com` top-level domain represents 41.5% of the vulnerabilities found, but the whole crawl consists of approximately 50% `.com` domains and therefore serves as an indication that owners of `.com` domains take better care of their application's security. Another example of this would be `.de` domains that represent 7.0% of all domains scanned but are only 5.9% of the vulnerabilities detected. On the other hand, TLDs such as `.it` and `.pl` have a significantly larger amount of vulnerabilities detected compared to sites scanned in the crawl. More examples of this can be seen in Figure 6.2, where the more red a country is on the map, the more it is over-represented in the vulnerabilities found in our scan.

Using the Tranco Top 1M list, we also looked at the most popular websites in the world and checked for any vulnerabilities detected on those domains. From the 1 million domains on the list, approximately 628 000 had been scanned by Common Crawl. Even though not all domains could be checked by our scan, as they had not been originally scanned by Common Crawl, the Tranco domains were still over-represented in our findings compared to other sites. Of these 1 million domains, indicators of vulnerability were detected on 9.3% of them (or 14,8% of the domains in the Common Crawl data). This is a significant increase from the ~10.9% detected on all domains scanned. The vulnerability indicators detected on these websites were similar to the general data, with the top 5 vulnerabilities detected being the same, albeit in different orders. Some of these vulnerabilities could be verified through our dynamic verification on multiple domains on the Tranco list. Some of the domains on the Tranco list are, however, domains that feature user-supplied web pages or

content, such as `Googleusercontent`. This domain, and others like it, might host user-created content on them that could be vulnerable, but that in itself does not indicate a vulnerability on the whole domain. Furthermore, as the crawler used by Common Crawl tries to mimic a real user, it would not be unreasonable for it to have scanned more pages from the top 1 million domain, thus giving more pages to find indicators of vulnerability on.

Through our dynamic analysis, we show that we can indeed verify the existence of certain vulnerabilities on the web applications that they were found on. Not all vulnerabilities that were tested could be verified, and this could be due to various reasons. Wrong fingerprinting information, stealth patches that had been performed, and verification blocked by Web Application Firewalls (WAFs) [45] or Content Security Policies (CSPs) [44] are some of them. By performing this final dynamic verification, we could finally answer the last research question (**RQ3**) of this thesis; *Can the found indicators of vulnerability be verified, e.g. by checking for false positives or running tests?*, with a "yes, but not for all vulnerabilities".

7.1 Vulnerability Selection

The vulnerability selection was done quite early on in the project. We focused on establishing a wide sample of different vulnerability types being found using a multitude of detection methods created for the data available to us. In hindsight, a few of the vulnerabilities selected had quite a bit of overlap, were hard to verify, or created substantial amounts of false positives, and the thesis would have benefited from a modified selection.

However, whilst some vulnerabilities proved to be problematic, the majority of the vulnerabilities still worked and demonstrated the success of this thesis. Due to the large variation in possible vulnerabilities, we managed to produce results that show that it is possible to detect a large variety of different vulnerabilities whilst utilizing many different detection and fingerprinting methods.

It can also be determined that the type of vulnerability did not matter regarding whether or not we could detect it. The only criterion that had an impact was whether a vulnerability could be fingerprinted successfully. Whilst a lot of traditional vulnerability detection methods utilize specific methods for different vulnerabilities, the fingerprinting method utilized in this thesis can be used for all types.

Some vulnerabilities were not detected at all in the data set from Common Crawl. These were two login portals and one PHP shell backdoor. The login portals are not supposed to be detected if implemented correctly to avoid exposing these to the web. However, it could also be that our way of seeing these specific login pages was flawed and thus missed in the scan.

The PHP shell that was not detected could either mean that the way our detection method worked was insufficient to detect it, our fingerprinting/comparison text was not up to date, or none of the web applications and websites scanned had this shell.

7.2 False Positives

When analyzing the results achieved from the large-scale scan, certain issues became apparent. It was noted that for certain specific vulnerabilities, the data gave way to quite a large number of false positives. Due to how fingerprinting works, especially for keyword searches, a lot of results match the correct search terms but are not in line with what we hoped to find. Examples of these are:

- SQL error messages
- IOC vulnerabilities (shells, backdoors, and error texts)
- Index of pages

For SQL error messages, we noted that we did get some correct results where a vulnerability could be detected from the error messages detected on certain web applications. We did, however, also note that a lot of the results obtained were forum posts, FAQs, or other pages talking about the error and how to solve it (e.g. Stack Overflow threads and similar forums).

Certain IOC vulnerabilities that were scanned for using keyword searches also returned some false positives. These ranged from pages discussing the vulnerabilities, how to handle them, and proof-of-concepts set up to showcase the vulnerabilities. For example, a search for a specific RCE error text (vulnerability ID 40) yielded results with text which indicates the possibility of a vulnerability. However, parts of the results came from security forums discussing how this can be seen as an indicator of vulnerability and were thus false positives.

Another search that can lead to a lot of possible false positives is the scan for ”Index of” pages (vulnerability ID 14). The files and information disclosed on these pages are not always indicators of bad practice or a possible vulnerability, as these might be documents that should be publicly available. However, these pages can lead to path traversal if not implemented correctly even if some of the data should be available. To determine which of these are false positives and which are not would require a lot of time and effort and can not necessarily be counted as vulnerabilities.

The conclusion from these results and observations is that when using plain text to find specific keywords on web applications, it is important to either verify the results or have more restrictive search criteria that eliminate these false positives. Another way to get around the issue is to use other detection methods to find these kinds of vulnerabilities. By searching for vulnerabilities in specific HTML tags such as `script` or `href`, better results should be able to be obtained. An example of this would be the Leafmailer email client which is an indicator of compromise. During the early stages of this thesis, we searched for this vulnerability through keywords in the WET files but noted that we got some false positives. We then evaluated the fingerprinting and noted that pages with the Leafmailer client also contained a specific script tag that could be used to fingerprint it. Whilst this might not be possible for all IOCs examined in this thesis, it could still help reduce the number of false positives for certain IOCs scanned for. Another way to avoid false positives could be to use multiple criteria to verify a specific vulnerability. This was done in

the case of the WordPress Enfold Theme, where the vulnerability required a second plugin (Avia Pagination) to be exploitable. By searching just for the vulnerable Enfold version, we found drastically more web applications than when searching using multiple criteria. Whilst the recommendation would still be to upgrade the Enfold version even if Avia Pagination is not used, the indication of vulnerability is much lower, and thus the number of false positives could be reduced for other vulnerabilities by utilizing a similar approach.

7.3 Comparison to Existing Resources

There already exists some similar services to what this thesis aims to produce. Two of those are Shodan and PublicWWW, and both have some of the same capabilities. Shodan allows for searches for specific vulnerabilities as well as libraries, IP addresses, and much more, whereas PublicWWW only allows for searches for specific libraries or services.

Table 7.1: Comparison with similar services. * implies that, while our scanner is not a paid service, there are still operating costs associated with it. ** indicates that whilst the scans are not performed when searched for, a user can request new (more up-to-date) scans on the selected domains.

Name	CC Scanner	Shodan	PublicWWW
Website count	3.1B	-	482M
Domain Count	40M	600M	-
Paid	No*	Yes	Yes
Search using CVEs	Yes	Yes	No
Dynamic scan	No	Yes**	No

A lot of the same functionality exists in all services where it is possible to search for different libraries. Since Shodan is aimed towards cybersecurity, it is also possible to search for certain CVEs as well, whilst PublicWWW requires specifying what library and version that wants to be searched for due to PublicWWW not necessarily focusing on fingerprinting for security purposes. The CVE search that is possible on Shodan is however quite limited as it requires Shodan to have performed a fingerprint match between the website and the vulnerability searched for. As can be seen in the results this is not always the case, and thus the CVE search can be unreliable. Furthermore, the number of results obtained by Shodan might not be quite accurate since it includes duplicates due to Shodan scanning multiple ports on the same IP address. This leads to both the HTTP and HTTPS sites being scanned thus giving duplicate results. As not all websites use HTTP or HTTPS, it can be hard to determine which number would be more accurate and thus the total as presented by Shodan was displayed.

Shodan is a paid service where the domains/IPs that should be monitored can be specified by a paying user. This means that the number of websites scanned might be lower, as their URL count is not specified in their documentation, compared to

both this thesis and PublicWWW. All of these services use indexed data and do not get the data on demand with the exception of Shodan, where a paying user can request more up-to-date scans using scan credits.

A comparison of a few vulnerabilities fingerprinted and detected in this thesis compared to PublicWWW and Shodan was performed in Table 6.8. As seen in the table, there is a mixed bag of results. We were able to detect more domains for some of the vulnerabilities, and for others, we detected fewer domains. Furthermore, some vulnerabilities cannot be detected on Shodan using their `vuln` search feature, most likely due to not being able to match any fingerprinting to the vulnerability, but could instead be found using Shodan's free text search. For PublicWWW it was not possible to determine the number of vulnerable websites for all vulnerabilities as PublicWWW only supports searching for all versions of a library/system or only 1 specific version. Thus it would be infeasible to search for some vulnerabilities as there are too many vulnerable versions. By allowing the search for specific CVEs either using the `vuln` tag (Shodan) or being able to specify multiple versions and/or criteria (our scanner) a service becomes much more user-friendly compared to having to perform multiple searches for different versions of the same vulnerability.

7.4 Ethical Considerations

In this thesis, multiple areas need to be carefully thought through in order to work ethically on this project. As we in this thesis looked at vulnerabilities and vulnerable web applications, it was important to not cause any disruptions to the websites looked at. As most of the heavy analysis was performed on static data, no additional stress on the websites was created. Furthermore, not all websites with indicators of compromise were dynamically tested.

7.4.1 Ethically verifying

When dynamically testing the vulnerabilities that had been found in the data from the Common Crawl scans, it was important that we would not cause any disruption or alterations to the applications tested or their users. Thus it would be impossible to perform verification for all vulnerabilities that we scanned for as some would be intrusive or damaging.

The payloads used in this thesis were either client-side alterations such as popups, shell `echo` commands, or payloads with a similar effect that would not change anything on the server side. By running these kinds of verification tests, a chosen set of vulnerabilities could be dynamically verified.

Certain vulnerabilities would have required verifications such as buffer overflow, denial of service, or request smuggling to verify and could affect either the whole server or other users and were thus not performed in this thesis.

7.4.2 Disclosing Code

Throughout the project, we have been unsure of how to handle the source code that has been created. Due to working with security vulnerabilities, we need to think extra hard about what kind of impact this thesis can have. In this thesis, we aim to provide data and methodology for examining vulnerabilities on a very large scale. This could be used by companies trying to get information on how many web applications are running their vulnerable code or to warn the hosts of these applications that they might have vulnerabilities.

Whilst this can have a lot of positive impacts by highlighting security risks or making hosts more aware of potential threats, it can also work as a way for a malicious user to find a multitude of applications to run boilerplate code on. Instead of a malicious actor having to manually find the websites that have a specific vulnerability, they could utilize a version of the code used in this project to quite quickly and effectively find hundreds or thousands of domains that could be vulnerable and attacked all at once.

A way to get around the possibility of malicious use could be to not release the source code but make the project available as a service that would have to be signed up for. By verifying the users, some (but not all) malicious actors could be stopped from using the service. Other than just user verification, the results could be returned in an obfuscated format or a rate limit on how much information can be received. This would still allow for statistical analysis but would limit the impact on individual applications.

As this thesis is based on the Common Crawl data obtained through their parsing, the rate at which the data is received is quite delayed as new crawls are only released every other month. This would give some leeway that would allow hosts to update their applications before the data becomes available.

7.4.3 Disclosing Vulnerabilities

In order to avoid this thesis being used to find websites/web applications that are vulnerable, no actual websites with vulnerabilities are referenced, and rather the statistics for each vulnerability are. By doing this we still manage to provide insight into whether or not it is possible to perform these kinds of large-scale vulnerability analysis without compromising the security of any of the websites that we discover.

If we would want to release information regarding any specific websites we would first want to do some kind of responsible disclosure to the companies/owners of the websites so that they would have the opportunity to fix the vulnerabilities before they would become public knowledge. But since that is not currently necessary and due to the vast number of vulnerable websites, responsible disclosure to all of them would not be possible.

7.5 Large-scale Data Takeaways

Working with data sets of this size has been challenging. Whilst it has been difficult to create a cost-effective and accurate solution to the problem, it was still manageable after a couple of lessons had been learned. The programs and resources you use to complete the project really matter in these scenarios. If not for the extra computational power provided by the AWS Lambda instances, parts of this thesis would have taken a much longer time or been completely infeasible. However, working with these resources bring new issues that have to be resolved.

7.5.1 Idempotentcy

Running a multitude of simultaneous work requires us to always check that we do not run the same calculations multiple times and thus duplicate results and data. After running a scan on 10%, we noticed that 33 segments had been processed two times. This occurs when "one of the servers which store a copy of the message is unavailable when you send the request to delete the message"[1][2]. Therefore, the messages would not be deleted, and after the visibility timeout, be picked up by a new Lambda instance. The recommendation from AWS is that one configures the Lambda to be idempotent. That is, if the same message is received twice the result should only be saved once. We then configured the vulnerability scanner to check the statistics database if a segment had already been processed before processing each segment and thus introducing idempotency into our solution.

7.5.2 Choosing the Correct Tools

During the scan, we noticed very few issues with the tools and resources we chose to use early on in the project. However, once we got into the analysis part of our thesis, it quickly became apparent that not all tools worked as we had expected them to. One of these resources we had issues with was the DocumentDB database. A lot of the functionality that we would normally expect from a MongoDB instance was not implemented into DocumentDB as it is not a genuine MongoDB instance. Instead of utilizing some functions that would normally be available to analyze the data such as Map-Reduce or certain aggregation methods, had to be done manually by first extracting the data and running these queries locally before uploading the data to the database again.

7.5.3 Working with Common Crawl data

Working with the data was a tricky experience, largely due to the large size of the data set provided by Common Crawl. Another aspect that could have been considered was the use of other file formats provided by Common Crawl to easily get certain data such as web-graphs, to get all URLs with links to certain domains.

[1] `https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_DeleteMessage.html`

[2] `https://repost.aws/knowledge-center/lambda-function-process-sqs-messages`

Due to the data being openly available to AWS users, we also experienced issues with rate limits as other users were accessing the same files at the same time. This meant that duplicate runs on the same segment had to be performed. However, due to the retry behavior and the idempotency we built in, this turned out to be a solvable issue.

7.6 Future work

There are several ways in which one could expand on the work presented in this thesis. To get an even better understanding of the vulnerabilities that could be found, other data sets could be used, or a purpose-built crawler could be developed to have better control over what websites and domains that should be checked.

Further improvements that could be made would be to either increase the number of vulnerabilities scanned for in order to get an overall better picture of the state of vulnerabilities on a large scale or reduce the number of vulnerabilities and instead perform more extensive searches with better accuracy and lower false positive rates. This could help showcase how any specific vulnerability could be analyzed. This could be extended further by looking at multiple crawls performed by Common Crawl. By looking at various time periods, one could gather a deeper understanding regarding how long vulnerabilities are present after disclosure and thus find interesting update patterns for these vulnerabilities.

Another type of scan that could be run that could gather further insight would be to check how many web applications are running a vulnerable version versus how many are running up-to-date or safe versions. By gathering information about this, we would be able to more accurately see the impact of certain vulnerabilities and the adoption rates for newer and safer versions of different software used.

The technical solution in itself could also be improved by utilizing or writing new libraries for various parts such as reading the files, getting the data from AWS S3, deciphering versions, and detecting vulnerabilities. By improving any of these parts, the solution could work faster and more accurately detect vulnerabilities.

Another improvement would be regarding vulnerabilities detected through keywords on websites. By improving this, the results obtained for the vulnerabilities through keyword searches would become more accurate and with a lower rate of false positives. This could be done by either extending the search phrases to more exactly fit “just” the intended result or introducing further criteria to increase accuracy.

8

Conclusion

This thesis shows that a large-scale analysis of known vulnerabilities can be done using datasets such as Common Crawl through cloud computation. We had the following research questions, which were all answered with a yes.

RQ1 Can vulnerabilities be detected from the Common Crawl data?

RQ2 Is it possible to scan for vulnerabilities on a very large scale using the Common Crawl data?

RQ3 Can the found indicators of vulnerability be verified, e.g. by checking for false positives or running tests?

By looking at the data presented in the different formats from Common Crawl, multiple ways have been found to detect vulnerabilities. This could range from pure text searches that can be used to find indicators of compromise or SQL errors that highlight the possibility for SQL injections to more specific HTML elements such as meta tags, scripts, and links being checked. It has been determined that the actual type of vulnerability does not matter for this project, but rather that the limitation of detecting possible vulnerabilities is related to how one can fingerprint different libraries and versions.

After finding ways to fingerprint and detect various libraries with known vulnerabilities, a proof of concept showed that we could extract and analyze the data effectively. We could then build upon and optimize this proof of concept to work with AWS and cloud computing not just to examine a small sample but rather a whole internet archive snapshot that Common Crawl stores. For this, a set of vulnerabilities and detection methods were run on the entire snapshot, and millions of websites with indicators of vulnerabilities were found for this minimal vulnerability sample.

Whilst millions of vulnerability indicators being found are auspicious, it is still important to validate how accurate the data gathered is. To check this, a dynamic analysis was performed. By first validating that the vulnerability indicator still exists and then performing non-intrusive checks, we could verify the validity of the results. However, not all vulnerabilities were verified since some would require intrusive or disruptive checks. While we did not perform intrusive checks, we do not see any significant technical challenges in testing most of these. From this dynamic analysis, we could see that the indicators collected do not match perfectly. Since the crawl, some websites had been updated to new versions, some websites could no longer be

reached, some websites did not run the vulnerable parts of the code, and some had countermeasures to stop our checks. Regardless, many could be verified, which shows that a large-scale vulnerability analysis on the HTTP responses in the Common Crawl data can be performed.

Bibliography

[1] *CVE Program homepage*. [Online]. Available: https://www.cve.org/ (visited on 12/03/2022).

[2] *Security week heartbleed still affects 200,000 devices: Shodan*. [Online]. Available: https://www.securityweek.com/heartbleed-still-affects-200000-devices-shodan (visited on 12/05/2022).

[3] *CommonCrawl homepage*. [Online]. Available: https://commoncrawl.org/ (visited on 12/03/2022).

[4] *Common crawl - november/december 2022 crawl archive now available*. [Online]. Available: https://commoncrawl.org/2022/12/nov-dec-2022-crawl-archive-now-available/ (visited on 02/03/2022).

[5] AWS Editorial Team, *Analyzing performance and cost of large-scale data processing with aws lambda*. [Online]. Available: https://aws.amazon.com/blogs/apn/analyzing-performance-and-cost-of-large-scale-data-processing-with-aws-lambda/ (visited on 05/10/2023).

[6] *What is shodan*. [Online]. Available: https://help.shodan.io/the-basics/what-is-shodan (visited on 05/11/2023).

[7] *Publicwww homepage*. [Online]. Available: https://publicwww.com/ (visited on 01/24/2023).

[8] *Owasp web security testing guide*. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/ (visited on 01/25/2023).

[9] *January/february 2023 crawl archive now available – common crawl*. [Online]. Available: https://commoncrawl.org/2023/02/jan-feb-2023-crawl-archive-now-available/ (visited on 03/23/2023).

[10] S. Nagel, *commoncrawl vs archive.org etc | Google Groups discussion*. [Online]. Available: https://groups.google.com/g/common-crawl/c/RBFAn0o55cY/m/68qiLwZMBAAJ (visited on 05/30/2023).

[11] *Frequently asked questions – common crawl*. [Online]. Available: https://commoncrawl.org/big-picture/frequently-asked-questions/ (visited on 03/24/2023).

[12] *Sebastian Nagel response to how the common crawl crawler works*. [Online]. Available: https://groups.google.com/g/common-crawl/c/sNe1nsUFawg/m/niNg1O-LBwAJ (visited on 05/15/2023).

[13] S. Nagel, *Response regarding how common crawl decides which websites to visit*. [Online]. Available: https://groups.google.com/g/common-crawl/c/6iTPnQrtGLs/m/4bCw6HNzCAAJ.

[14] IIPC, "The WARC Format," International Organization for Standardization, Standard ISO 28500:2017, 2017. [Online]. Available: https://iipc.github.io/warc-specifications/specifications/warc-format/warc-1.1/.

[15] *CommonCrawl get started guide*. [Online]. Available: https://aws.amazon.com/blogs/apn/analyzing-performance-and-cost-of-large-scale-data-processing-with-aws-lambda/ (visited on 05/17/2023).

[16] R. White, "Namibia braces for nujoma exit," *BBC News*, Jan. 22, 2004. [Online]. Available: http://news.bbc.co.uk/2/hi/africa/3414345.stm (visited on 05/15/2023).

[17] K. Dempsey, E. Takamura, P. Eavy, and G. Moore, *Automation support for security control assessments:* Apr. 2020, p. 93. DOI: 10.6028/nist.ir.8011-4. [Online]. Available: http://dx.doi.org/10.6028/NIST.IR.8011-4.

[18] A. Hoffman, *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*, 1st ed. O'Reilly Media, 2020, ISBN: 9781492053118.

[19] J. Kahlin and I. L. Valbuena, *File:persistent-xss.png – excess xss*. [Online]. Available: https://excess-xss.com/ (visited on 03/28/2023).

[20] P. Yaworski, *Real-world bug hunting*. San Francisco, CA: No Starch Press, Jul. 2019.

[21] *What is prototype pollution? | Tutorial & examples | Snyk Learn*. [Online]. Available: https://learn.snyk.io/lessons/prototype-pollution/javascript/ (visited on 03/28/2023).

[22] G. Najera-Gutierrez and J. A. Ahmed, *Web Penetration Testing with Kali Linux*, Third Edition. Packt Publishing, 2018, pp. 346–349, ISBN: 9781788623377.

[23] OWASP, *Owasp web security testing guide - test upload of unexpected file types*. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/10-Business_Logic_Testing/08-Test_Upload_of_Unexpected_File_Types (visited on 03/28/2023).

[24] *Unrestricted File Upload*. [Online]. Available: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload (visited on 05/26/2023).

[25] OWASP, *Owasp web security testing guide - testing for privilege escalation*. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/05-Authorization_Testing/03-Testing_for_Privilege_Escalation (visited on 03/28/2023).

[26] Fortinet, *Indicators of compromise (iocs)*. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/indicators-of-compromise (visited on 03/28/2023).

[27] OWASP, *Owasp web security testing guide - review webpage content for information leakage*. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/05-Review_Webpage_Content_for_Information_Leakage (visited on 03/28/2023).

[28] *What is AWS*. [Online]. Available: https://aws.amazon.com/what-is-aws/ (visited on 05/15/2023).

[29] *Serverless Computing - AWS Lambda - Amazon Web Services*. [Online]. Available: https://aws.amazon.com/lambda/ (visited on 12/03/2022).

[30] *Fully Managed Message Queuing – Amazon Simple Queue Service – Amazon Web Services*. [Online]. Available: https://aws.amazon.com/sqs/ (visited on 05/15/2022).

[31] *Cloud Object Storage – Amazon S3 – Amazon Web Services*. [Online]. Available: https://aws.amazon.com/s3/ (visited on 05/15/2022).

[32] *Amazon DocumentDB*. [Online]. Available: https://aws.amazon.com/documentdb/ (visited on 05/15/2022).

[33] *Secure and resizable cloud compute – Amazon EC2 – Amazon Web Services*. [Online]. Available: https://aws.amazon.com/ec2/ (visited on 05/15/2022).

[34] *Amazon VPC*. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html (visited on 05/15/2022).

[35] MDN contributors, *<meta>: The metadata element - HTML: HyperText Markup Language | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta (visited on 05/17/2023).

[36] MDN contributors, *<script>: The Script element - HTML: HyperText Markup Language | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script (visited on 05/17/2023).

[37] MDN contributors, *HTTP headers - HTTP | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (visited on 05/17/2023).

[38] MDN contributors, *<title>: The Document Title element - HTML: HyperText Markup Language | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title (visited on 05/17/2023).

[39] *OWASP Review Webpage Comments and Metadata for Information Leakage*. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/01-Information_Gathering/05-Review_Webpage_Comments_and_Metadata_for_Information_Leakage (visited on 12/03/2022).

[40] OWASP, *Improper error handling | owasp foundation*. [Online]. Available: https://owasp.org/www-community/Improper_Error_Handling (visited on 06/21/2023).

[41] *Owasp top ten*. [Online]. Available: https://owasp.org/www-project-top-ten/ (visited on 12/03/2022).

[42] *CVE-2020-17496*. Available from MITRE, CVE-ID CVE-2020-17496. Aug. 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-17496 (visited on 05/15/2023).

[43] A. Casalboni, *AWS Lambda Power Tuning*. [Online]. Available: https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:451282441545:applications~aws-lambda-power-tuning (visited on 05/22/2023).

[44] MDN contributors, *Content Security Policy (CSP) - HTTP | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP (visited on 05/23/2023).

[45] Cloudflare, *What is a WAF? | Web Application Firewall explained | Cloudflare*. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/ (visited on 05/23/2023).

[46] *Tranco homepage*. [Online]. Available: `https://tranco-list.eu/` (visited on 12/05/2022).

[47] *Web servers technologies market share*. [Online]. Available: `https://www.wappalyzer.com/technologies/web-servers/` (visited on 05/22/2023).

A

WAT Record Example

```
WARC/1.0
WARC-Type: metadata
WARC-Target-URI:
↪ https://wiki.p2pfoundation.net/Cooperative_Intelligence
WARC-Date: 2023-02-09T17:42:45Z
WARC-Record-ID: <urn:uuid:e49f745c-2e1d-4fff-938f-7f83bb286fd0>
WARC-Refers-To: <urn:uuid:6730aae8-d5d0-43b9-8477-fb7892c56db2>
Content-Type: application/json
Content-Length: 8863

{
    "Container": {
        "Filename":
↪ "CC-MAIN-20230126210844-20230127000844-00000.warc.gz",
        "Compressed": true,
        "Offset": "628984542",
        "Gzip-Metadata": {
            "Deflate-Length": "6633",
            "Header-Length": "10",
            "Footer-Length": "8",
            "Inflated-CRC": "277398458",
            "Inflated-Length": "18623"
        }
    },
    "Envelope": {
        "Payload-Metadata": {
            "Actual-Content-Type": "application/http;
↪ msgtype=response",
            "HTTP-Response-Metadata": {
                "Response-Message": {
                    "Status": "200",
                    "Version": "HTTP/1.1",
                    "Reason": "OK"
                },
                "Headers-Length": "939",
                "Headers": {
```

```
                    "Date": "Thu, 26 Jan 2023 21:30:55 GMT",
                    "Content-Type": "text/html; charset=UTF-8",
                    "X-Crawler-Transfer-Encoding": "chunked",
                    "Connection": "keep-alive",
                    "X-Content-Type-Options": "nosniff",
                    "Vary": "Accept-Encoding,Cookie",
                    "Expires": "Thu, 01 Jan 1970 00:00:00 GMT",
                    "Cache-Control": "private, must-revalidate,
↪ max-age=0",
                    "Last-Modified": "Mon, 30 Nov 2020 13:24:11 GMT",
                    "Content-Language": "en",
                    "X-Request-Duration": "D=535029",
                    "CF-Cache-Status": "DYNAMIC",
                    "Report-To": "{\"endpoints\":[{\"url\":\"https:\
↪ \/\\\/a.nel.cloudflare.com\\/report\\/v3?s=NGUFZsnwlNUsyDoHcaN%2F
↪ QYuNtYF%2Bki2YmxL2ZC6FPDx%2Fkud03PuNnr2uq2O3SJpZLcCgjvAsSnKF%2B6
↪ 9K2s2RaAGi7UlQD%2B%2FFy%2F9Zm6Xo%2F0Icpkzai%2Fd50QLLRR1nK8Uahc0n
↪ iDk4Zjs4\"}],\"group\":\"cf-nel\",\"max_age\":604800}",
                    "NEL": "{\"success_fraction\":0,\"report_to\":\"
↪ cf-nel\",\"max_age\":604800}",
                    "Server": "cloudflare",
                    "CF-RAY": "78fc5a3929da242d-IAD",
                    "X-Crawler-Content-Encoding": "br",
                    "alt-svc": "h3=\":443\"; ma=86400,
↪ h3-29=\":443\"; ma=86400",
                    "Content-Length": "17074"
                },
                "HTML-Metadata": {
                    "Head": {
                        "Title": "Cooperative Intelligence - P2P
↪ Foundation",
                        "Link": [
                            {
                                "path": "LINK@/href",
                                "url": "/load.php?lang=en&modules=me
↪ diawiki.legacy.commonPrint%2Cshared%7Cmediawiki.skinning.interfa
↪ ce%7Cskins.vector.styles&only=styles&skin=vector",
                                "rel": "stylesheet"
                            },
                            {
                                "path": "LINK@/href",
                                "url":
↪ "/load.php?lang=en&modules=site.styles&only=styles&skin=vector",
                                "rel": "stylesheet"
                            },
                            {
```

```
                                "path": "LINK@/href",
                                "url": "/favicon.ico",
                                "rel": "shortcut icon"
                            },
                            {
                                "path": "LINK@/href",
                                "url": "/opensearch_desc.php",
                                "rel": "search",
                                "type":
↪   "application/opensearchdescription+xml"
                            },
                            {
                                "path": "LINK@/href",
                                "url":
↪   "//wiki.p2pfoundation.net/api.php?action=rsd",
                                "rel": "EditURI",
                                "type": "application/rsd+xml"
                            },
                            {
                                "path": "LINK@/href",
                                "url": "/P2P_Foundation:Copyright",
                                "rel": "license"
                            },
                            {
                                "path": "LINK@/href",
                                "url":
↪   "/index.php?title=Special:RecentChanges&feed=atom",
                                "rel": "alternate",
                                "type": "application/atom+xml"
                            }
                        ],
                        "Scripts": [
                            {
                                "path": "SCRIPT@/src",
                                "url": "/load.php?lang=en&modules=st
↪   artup&only=scripts&raw=1&skin=vector"
                            },
                            {
                                "path": "SCRIPT@/src",
                                "url":
↪   "https://analytics.example.com/tracking.js",
                                "type": "text/javascript"
                            }
                        ],
                        "Metas": [
                            {
```

```
                                "name":
↪  "ResourceLoaderDynamicStyles",
                                "content": ""
                            },
                            {
                                "name": "generator",
                                "content": "MediaWiki 1.34.1"
                            }
                        ]
                    },
                    "Links": [
                        {
                            "path": "A@/href",
                            "url": "#mw-head",
                            "text": "Jump to navigation"
                        },
                        {
                            "path": "A@/href",
                            "url": "#p-search",
                            "text": "Jump to search"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "http://www.cooperativeintelligence.org/",
                            "text":
↪  "http://www.cooperativeintelligence.org/"
                        },
                        {
                            "path": "A@/href",
                            "url": "https://wiki.p2pfoundation.net/i
↪  ndex.php?title=Cooperative_Intelligence&oldid=46793",
                            "text": "https://wiki.p2pfoundation.net/
↪  index.php?title=Cooperative_Intelligence&oldid=46793"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Special:Categories",
                            "title": "Special:Categories",
                            "text": "Categories"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Category:Resources",
                            "title": "Category:Resources",
                            "text": "Resources"
```

```
                        },
                        {
                            "path": "A@/href",
                            "url": "/Category:Sharing",
                            "title": "Category:Sharing",
                            "text": "Sharing"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Category:Cooperation",
                            "title": "Category:Cooperation",
                            "text": "Cooperation"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Category:Intelligence",
                            "title": "Category:Intelligence",
                            "text": "Intelligence"
                        },
                        {
                            "path": "A@/href",
                            "url": "/index.php?title=Special:UserLog
↪  in&returnto=Cooperative+Intelligence",
                            "title": "You are encouraged to log in;
↪  however, it is not mandatory [o]",
                            "text": "Log in"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Special:RequestAccount",
                            "title": "You are encouraged to create
↪  an account and log in; however, it is not mandatory",
                            "text": "Request account"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Cooperative_Intelligence",
                            "title": "View the content page [c]",
                            "text": "Page"
                        },
                        {
                            "path": "A@/href",
                            "url": "/index.php?title=Talk:Cooperativ
↪  e_Intelligence&action=edit&redlink=1",
                            "title": "Discussion about the content
↪  page (page does not exist) [t]",
```

```
                        "rel": "discussion",
                        "text": "Discussion"
                    },
                    {
                        "path": "A@/href",
                        "url": "/Cooperative_Intelligence",
                        "text": "Read"
                    },
                    {
                        "path": "A@/href",
                        "url":
↪  "/index.php?title=Cooperative_Intelligence&action=edit",
                        "title": "This page is protected.\nYou
↪  can view its source [e]",
                        "text": "View source"
                    },
                    {
                        "path": "A@/href",
                        "url":
↪  "/index.php?title=Cooperative_Intelligence&action=history",
                        "title": "Past revisions of this page
↪  [h]",
                        "text": "View history"
                    },
                    {
                        "path": "FORM@/action",
                        "url": "/index.php"
                    },
                    {
                        "path": "A@/href",
                        "url": "/Main_Page",
                        "title": "Visit the main page"
                    },
                    {
                        "path": "A@/href",
                        "url": "http://commonstransition.org/",
                        "text": "Commons Transition"
                    },
                    {
                        "path": "A@/href",
                        "url": "/Category:Open_Cooperativism",
                        "text": "Open Cooperativism"
                    },
                    {
                        "path": "A@/href",
                        "url": "http://www.p2plab.gr/en/",
```

```
                        "text": "P2P Lab"
                    },
                    {
                        "path": "A@/href",
                        "url": "/Main_Page",
                        "text": "Wiki Homepage"
                    },
                    {
                        "path": "A@/href",
                        "url": "https://blog.p2pfoundation.net/",
                        "text": "International Blog [EN]"
                    },
                    {
                        "path": "A@/href",
                        "url":
↪   "http://p2pfoundation.net/Greek_language",
                        "text": "Greek Language [GR]"
                    },
                    {
                        "path": "A@/href",
                        "url":
↪   "https://wikifr.p2pfoundation.net",
                        "text": "French Language [FR]"
                    },
                    {
                        "path": "A@/href",
                        "url": "http://blognl.p2pfoundation.net",
                        "text": "Dutch Language Blog [BE-NL]"
                    },
                    {
                        "path": "A@/href",
                        "url": "/P2P_Foundation_Email_Lists",
                        "text": "Mailing List"
                    },
                    {
                        "path": "A@/href",
                        "url": "/Special:MultiCategorySearch",
                        "text": "Multi-Category Search"
                    },
                    {
                        "path": "A@/href",
                        "url": "/Special:RecentChanges",
                        "title": "A list of recent changes in
↪   the wiki [r]",
                        "text": "Recent changes"
                    },
```

```
                        {
                            "path": "A@/href",
                            "url": "/Special:Random",
                            "title": "Load a random page [x]",
                            "text": "Random page"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Help:Contents",
                            "title": "The place to find out",
                            "text": "Help"
                        },
                        {
                            "path": "A@/href",
                            "url": "/P2P_Foundation:Site_support",
                            "text": "Donations"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "/Special:WhatLinksHere/Cooperative_Intelligence",
                            "title": "A list of all wiki pages that
↪  link here [j]",
                            "text": "What links here"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "/Special:RecentChangesLinked/Cooperative_Intelligence",
                            "title": "Recent changes in pages linked
↪  from this page [k]",
                            "rel": "nofollow",
                            "text": "Related changes"
                        },
                        {
                            "path": "A@/href",
                            "url": "/Special:SpecialPages",
                            "title": "A list of all special pages
↪  [q]",
                            "text": "Special pages"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "/index.php?title=Cooperative_Intelligence&printable=yes",
```

```
                            "title": "Printable version of this page
↪  [p]",
                            "rel": "alternate",
                            "text": "Printable version"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "/index.php?title=Cooperative_Intelligence&oldid=46793",
                            "title": "Permanent link to this
↪  revision of the page",
                            "text": "Permanent link"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "/index.php?title=Cooperative_Intelligence&action=info",
                            "title": "More information about this
↪  page",
                            "text": "Page information"
                        },
                        {
                            "path": "A@/href",
                            "url": "/index.php?title=Special:CiteThi
↪  sPage&page=Cooperative_Intelligence&id=46793",
                            "title": "Information on how to cite
↪  this page",
                            "text": "Cite this page"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "/Special:CreateRedirect/Cooperative_Intelligence",
                            "text": "Create Redirect"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "http://p2pfoundation.net/P2P_Foundation:Copyright",
                            "title": "P2P Foundation Notice on
↪  Copyrights",
                            "text": "Copyright Information"
                        },
                        {
                            "path": "A@/href",
                            "url": "/P2P_Foundation:Privacy_policy",
```

```
                            "title": "P2P Foundation:Privacy policy",
                            "text": "Privacy policy"
                        },
                        {
                            "path": "A@/href",
                            "url": "/P2P_Foundation:About",
                            "title": "P2P Foundation:About",
                            "text": "About the P2P Foundation Wiki"
                        },
                        {
                            "path": "IMG@/src",
                            "url":
↪  "https://i.creativecommons.org/l/by-sa/3.0/88x31.png",
                            "alt": "Attribution-ShareAlike 3.0
↪  Unported"
                        },
                        {
                            "path": "A@/href",
                            "url":
↪  "https://creativecommons.org/licenses/by-sa/3.0/"
                        },
                        {
                            "path": "IMG@/src",
                            "url":
↪  "/resources/assets/poweredby_mediawiki_88x31.png",
                            "alt": "Powered by MediaWiki"
                        },
                        {
                            "path": "A@/href",
                            "url": "https://www.mediawiki.org/"
                        }
                    ]
                },
                "Entity-Length": "17074",
                "Entity-Digest":
↪  "sha1:INLWAV4CUJZJZFBH4PZ6W42EH6Q5U3JL",
                "Entity-Trailing-Slop-Length": "0"
            },
            "Actual-Content-Length": "18013",
            "Trailing-Slop-Length": "4",
            "Block-Digest": "sha1:TX5GDCD22YN4GXQBMJHGRGRXOFWMGO2E"
        },
        "Format": "WARC/1.0",
        "WARC-Header-Length": "606",
        "WARC-Header-Metadata": {
            "WARC-Type": "response",
```

```
            "WARC-Date": "2023-01-26T21:30:55Z",
            "WARC-Record-ID":
↪   "<urn:uuid:6730aae8-d5d0-43b9-8477-fb7892c56db2>",
            "Content-Length": "18013",
            "Content-Type": "application/http; msgtype=response",
            "WARC-Warcinfo-ID":
↪   "<urn:uuid:c3e66540-e545-4a14-a4aa-e6d245cbf6fd>",
            "WARC-Concurrent-To":
↪   "<urn:uuid:6eb6f890-5c02-45c6-90d9-ff6407002f5a>",
            "WARC-IP-Address": "172.67.208.141",
            "WARC-Target-URI":
↪   "https://wiki.p2pfoundation.net/Cooperative_Intelligence",
            "WARC-Payload-Digest":
↪   "sha1:INLWAV4CUJZJZFBH4PZ6W42EH6Q5U3JL",
            "WARC-Block-Digest":
↪   "sha1:TX5GDCD22YN4GXQBMJHGRGRXOFWMGO2E",
            "WARC-Identified-Payload-Type": "text/html"
        }
    }
}
```

B

WET Record Example

```
WARC/1.0
WARC-Type: conversion
WARC-Target-URI:
↪ http://aermp.org/risk-alert-newsletter/what-is-enterprise-risk-management/
WARC-Date: 2023-01-26T22:49:52Z
WARC-Record-ID: <urn:uuid:04eb0cf4-c8e7-47bc-a698-3f1ba858f5f0>
WARC-Refers-To: <urn:uuid:ceacd17b-eca1-4a0c-9d4f-2e2aa6c73b17>
WARC-Block-Digest: sha1:ZUPVAUV37FNOEI7WT2AIIA7VXRQJF3HH
WARC-Identified-Content-Language: eng
Content-Type: text/plain
Content-Length: 2576

WARC/1.0
WARC-Type: conversion
WARC-Target-URI: http://aermp.org/risk-alert-newsletter/what-is-ente
↪ rprise-risk-management/
WARC-Date: 2023-01-26T22:49:52Z
WARC-Record-ID: <urn:uuid:04eb0cf4-c8e7-47bc-a698-3f1ba858f5f0>
WARC-Refers-To: <urn:uuid:ceacd17b-eca1-4a0c-9d4f-2e2aa6c73b17>
WARC-Block-Digest: sha1:ZUPVAUV37FNOEI7WT2AIIA7VXRQJF3HH
WARC-Identified-Content-Language: eng
Content-Type: text/plain
Content-Length: 2576

What is Enterprise Risk Management - AERMP
Have any questions?
+234 706 220 5563
+234 704 622 6901
info@aermp.org
Home
About Us
Who We Are
Vision & Objectives
Our People
Risk Resources
Risk Alert Newsletter
```

```
Risk Education
Certification Programs
Become Certified
Blog
Members Area
Login
Registration
Member Profile
Dues Payment
Becoming a Member
Benefits
Careers
Media Centre
News
Events
Gallery
Contact
What is Enterprise Risk Management
Home
Blog
Risk Alert Newsletter
What is Enterprise Risk Management
Risk Management
April 4, 2013
AERMP partners US center for Risk Management
August 12, 2015
0
What is Enterprise Risk Management
COSO defines ERM as a process, effected by an entity board of
 ↪  directors, management and other personnel, applied in
 ↪  strategy-setting and across the enterprise, designed to identify
 ↪  potential events that may affect the entity, and manage risk to
 ↪  be within its risk appetite, to provide reasonable assurance
 ↪  regarding the achievement of entity objectives.
The implication of this definition can be summarised as follows:
A process, ongoing and flowing through an entity. Implication - ERM
 ↪  never ends. It's about an entity improvement of risk management
 ↪  capabilities and is a continuous journey, not a destination, for
 ↪  any organization that seeks to improve continually. Effected by
 ↪  people at every level of an organizationImplication Everyone
 ↪  should be involved in managing risk. Unfortunately in many
 ↪  organizations today, ERM is localised to a particular department
 ↪  (Risk Management Dept) and many officers are not even aware of
 ↪  the risks they are incurring on behalf of their organizations.
Thus, workers in establishments are divided into two as follows:
Those who incur risks
```

```
Those who are saddled with the responsibility of managing risks.
This must not be so in ERM compliant organizations. For your
 ↪  organization to be ERM compliant, think on these things.
Share this:
Click to share on Twitter (Opens in new window)
Click to share on Facebook (Opens in new window)
Like this:
Like Loading...
Related
Comments are closed.
About Us
AERMP is the foremost professional body for risk management practice
 ↪  in Nigeria.
We are an independent, not-for-profit institute, duly registered and
 ↪  approved by the Federal Government of Nigeria (under the Company
 ↪  and Allied Matters Act 1990) to set professional standards in
 ↪  enterprise risk management practice among the practitioners in
 ↪  all industries and sectors (both private and public).
Subscribe to Receive Updates
Name *
Email *
Sign Up
© 2022 AERMP. All Rights Reserved
%d bloggers like this:
```

C

Vulnerabilities Database entries


```
[{
  "_id": "1",
  "name": "WordPress WatchTowerHQ Plugin",
  "versionRanges": [
    {
      "versionTo": "3.6.16",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "WatchTowerHQ"
  ],
  "type": "PRIVILEGE_ESCALATION",
  "_class": "MetatagVulnerability",
  "link": "https://patchstack.com/database/vulnerability/watchtowerh
↪  q/wordpress-watchtowerhq-plugin-3-6-16-privilege-escalation",
  "active": true
},{
  "_id": "2",
  "name": "WordPress Core",
  "versionRanges": [
    {
      "versionTo": "6.0.2",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "WordPress"
  ],
  "type": "INFORMATION_LEAKAGE",
  "link": "https://patchstack.com/database/vulnerability/wordpress/w
↪  ordpress-core-6-0-2-data-exposure-vulnerability-via-rest-api",
  "_class": "MetatagVulnerability",
```

```
  "active": false
},{
  "_id": "3",
  "name": "Duraspace Dspace - Privilege escalation",
  "versionRanges": [
    {
      "versionTo": "7.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "Dspace"
  ],
  "type": "PRIVILEGE_ESCALATION",
  "_class": "MetatagVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2021-41189",
  "active": true
},{
  "_id": "4",
  "name": "Zblogcn Z-BlogPHP Open redirect",
  "versionRanges": [
    {
      "versionTo": "1.5.2",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "Z-BlogPHP"
  ],
  "type": "INFORMATION_LEAKAGE",
  "_class": "MetatagVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2020-18268",
  "active": true
},{
  "_id": "5",
  "name": "WordPress HTML Forms Plugin",
  "versionRanges": [
    {
      "versionTo": "1.3.24",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
```

```
    "HTML Forms"
  ],
  "type": "SQL_INJECTION",
  "_class": "MetatagVulnerability",
  "link": "https://patchstack.com/database/vulnerability/html-forms/
↪  wordpress-html-forms-plugin-1-3-24-auth-sql-injection-sqli-vulne
↪  rability",
  "active": true
},{
  "_id": "6",
  "name": "Google Dorks - Netgate pfSense Plus - Login",
  "text": [
    "Netgate pfSense Plus - Login"
  ],
  "type": "LOGIN_PORTAL",
  "_class": "TitleVulnerability",
  "link": "https://www.exploit-db.com/ghdb/8042",
  "active": true
},{
  "_id": "7",
  "name": "AngularJS - Prototype pollution",
  "versionRanges": [
    {
      "versionTo": "1.7.8",
      "versionFrom": "1.4.1",
      "comparisonType": "BET"
    }
  ],
  "text": [
    "angularjs"
  ],
  "type": "PROTOTYPE_POLLUTION",
  "_class": "ScriptVulnerability",
  "link": "https://security.snyk.io/vuln/SNYK-JS-ANGULAR-534884",
  "active": true
},{
  "_id": "8",
  "name": "General SQL errors",
  "text": [
    "You have an error in your SQL syntax",
    "conflict occurred in database",
    "right syntax to use near"
  ],
  "type": "SQL_INJECTION",
  "_class": "SQLVulnerability",
  "link": "",
```

```
  "active": true
},{
  "_id": "9",
  "name": "MSSQL errors",
  "text": [
    "Syntax error in SQL statement",
    "Ambiguous column name"
  ],
  "type": "SQL_INJECTION",
  "_class": "SQLVulnerability",
  "link": "",
  "active": true
},{
  "_id": "10",
  "name": "PostgreSQL errors",
  "text": [
    "Error: syntax error at or near"
  ],
  "type": "SQL_INJECTION",
  "_class": "SQLVulnerability",
  "link": "",
  "active": true
},{
  "_id": "12",
  "name": "WordPress Core - Path traversal",
  "versionRanges": [
    {
      "versionTo": "2.3.3",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    },
    {
      "versionTo": "2.5",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "WordPress"
  ],
  "type": "PATH_TRAVERSAL",
  "link": "https://www.cvedetails.com/cve/CVE-2008-4769/",
  "_class": "MetatagVulnerability",
  "active": true
},{
  "_id": "13",
```

```
  "name": "WordPress WPForms Pro Plugin",
  "versionRanges": [
    {
      "versionTo": "1.7.6",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "WPForms Pro"
  ],
  "type": "CSV_INJECTION",
  "_class": "MetatagVulnerability",
  "link":
↪  "https://patchstack.com/database/vulnerability/wpforms/wordpress
↪ -wpforms-pro-premium-plugin-1-7-6-csv-injection-vulnerability",
  "active": true
},{
  "_id": "14",
  "name": "Generic File Listing - index of",
  "text": [
    "index of "
  ],
  "type": "INFORMATION_LEAKAGE",
  "_class": "TitleVulnerability",
  "link": "https://cwe.mitre.org/data/definitions/548.html",
  "active": true
},{
  "_id": "15",
  "name": "WordPress Cryptocurrency Widgets Pack Plugin",
  "versionRanges": [
    {
      "versionTo": "1.8.1",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "Cryptocurrency Widgets"
  ],
  "type": "SQL_INJECTION",
  "_class": "MetatagVulnerability",
  "link": "https://patchstack.com/database/vulnerability/cryptocurre
↪  ncy-widgets-pack/wordpress-cryptocurrency-widgets-pack-plugin-1-
↪  8-1-sql-injection-sqli-vulnerability",
  "active": true
```

```
},{
  "_id": "17",
  "name": "WordPress Houzez Login Register Plugin",
  "versionRanges": [
    {
      "versionTo": "2.6.3",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "houzez/js/houzez_ajax_calls.js",
    "houzez/js/custom.js",
    "houzez/js/plugins.js"
  ],
  "type": "PRIVILEGE_ESCALATION",
  "_class": "ScriptVulnerability",
  "link": "https://patchstack.com/database/vulnerability/houzez-logi
↪  n-register/wordpress-houzez-login-register-plugin-2-6-3-privileg
↪  e-escalation",
  "active": true
},{
  "_id": "18",
  "name": "WordPress Zendrop - Global Dropshipping Plugin",
  "versionRanges": [
    {
      "versionTo": "1.0.0",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "Zendrop - Global Dropshipping"
  ],
  "type": "SQL_INJECTION",
  "_class": "MetatagVulnerability",
  "link": "https://patchstack.com/database/vulnerability/zendrop-dro
↪  pshipping-and-fulfillment/wordpress-zendrop-global-dropshipping-
↪  plugin-1-0-0-arbitrary-code-execution",
  "active": false
},{
  "_id": "19",
  "name": "WordPress 10Web Booster Plugin - Unauthenticated SQLi",
  "versionRanges": [
    {
      "versionTo": "2.12.23",
```

```
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "10Web Booster"
  ],
  "type": "SQL_INJECTION",
  "_class": "MetatagVulnerability",
  "link": "https://www.wordfence.com/threat-intel/vulnerabilities/wo
↪  rdpress-plugins/tenweb-speed-optimizer/10web-booster-website-spe
↪  ed-optimization-cache-page-speed-optimizer-21223-authenticated-s
↪  ql-injection",
  "active": false
},{
  "_id": "11",
  "name": "KILL_THE_NET indicator of compromise",
  "text": [
    "kill_the_net",
    "killthenet"
  ],
  "type": "IOC",
  "_class": "KeywordVulnerability",
  "link": "",
  "active": true
},{
  "_id": "21",
  "name": "WordPress Corsa Theme",
  "versionRanges": [
    {
      "versionTo": "1.5",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "Corsa"
  ],
  "type": "ARBITRARY_FILE_UPLOAD",
  "_class": "ScriptVulnerability",
  "link": "https://patchstack.com/database/vulnerability/corsa/wordp
↪  ress-corsa-theme-1-5-arbitrary-file-upload",
  "active": true
},{
  "_id": "22",
  "name": "WordPress Enfold Theme",
```

```
  "versionRanges": [
    {
      "versionTo": "4.8.1",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "themes/enfold"
  ],
  "type": "XSS",
  "_class": "ScriptVulnerability",
  "link": "https://www.exploit-db.com/exploits/50427",
  "active": true
},{
  "_id": "23",
  "name": "IOC website - Black Hat Ad Network",
  "text": [
    "violetlovelines.com",
    "specialblueitems.com",
    "greengoplatform.com",
    "findtrustclicks.com",
    "news.weatherplllatform.com",
    "check.statisticline.com"
  ],
  "type": "IOC",
  "_class": "ScriptVulnerability",
  "link": "https://blog.sucuri.net/2023/01/massive-campaign-uses-hac
↪  ked-wordpress-sites-as-platform-for-black-hat-ad-network.html",
  "active": true
},{
  "_id": "24",
  "name": "XSS JQuery UI",
  "versionRanges": [
    {
      "versionTo": "1.13.2",
      "versionFrom": "",
      "comparisonType": "LT"
    }
  ],
  "text": [
    "jquery-ui-core",
    "jquery/ui"
  ],
  "type": "XSS",
  "_class": "ScriptVulnerability",
```

```
  "link": "https://security.snyk.io/vuln/SNYK-JS-JQUERYUI-2946728",
  "active": true
},{
  "_id": "25",
  "name": "Apache HTTP Server Path Traversal/RCE",
  "versionRanges": [
    {
      "versionTo": "2.4.49",
      "versionFrom": "2.4.50",
      "comparisonType": "BET"
    }
  ],
  "text": [
    "Apache"
  ],
  "type": "RCE",
  "_class": "HeaderVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2021-41773",
  "active": true
},{
  "_id": "27",
  "name": "Google Dorks - Parallels User Portal",
  "text": [
    "Parallels User Portal"
  ],
  "type": "LOGIN_PORTAL",
  "_class": "TitleVulnerability",
  "link": "https://www.exploit-db.com/ghdb/8043",
  "active": true
},{
  "_id": "28",
  "name": "Vue JS - XSS",
  "versionRanges": [
    {
      "versionTo": "2.5.17",
      "versionFrom": "",
      "comparisonType": "LT"
    }
  ],
  "text": [
    "vue-js",
    "vue.js",
    "vue.min.js"
  ],
  "type": "XSS",
  "_class": "ScriptVulnerability",
```

```
  "link": "https://security.snyk.io/vuln/npm:vue:20180802",
  "active": true
},{
  "_id": "30",
  "name": "WordPress LearnPress Plugin <= 4.1.7.3.2 is vulnerable to
↪  SQL Injection",
  "versionRanges": [
    {
      "versionTo": "4.1.7.3.2",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "LearnPress"
  ],
  "type": "SQL_INJECTION",
  "_class": "ScriptVulnerability",
  "link": "https://patchstack.com/database/vulnerability/learnpress/
↪  wordpress-learnpress-wordpress-lms-plugin-plugin-4-1-7-3-2-sql-i
↪  njection",
  "active": true
},{
  "_id": "31",
  "name": "Fancy Product Designer plugin for WordPress is vulnerable
↪  to Cross-Site Request Forgery via the FPD_Admin_Import class
↪  that makes it possible for attackers to upload malicious files
↪  that could be used to gain webshell access to a server",
  "versionRanges": [
    {
      "versionTo": "4.7.5",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "FancyProductDesigner",
    "fancy-product-designer"
  ],
  "type": "CSRF",
  "_class": "ScriptVulnerability",
  "link": "https://www.cvedetails.com/cve/CVE-2021-4096/",
  "active": true
},{
  "_id": "32",
```

```
  "name": "ASP.NET Security Feature Bypass Vulnerability (Asp.net
↪  MVC)",
  "versionRanges": [
    {
      "versionTo": "5.2",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "X-AspNetMvc-Version"
  ],
  "type": "BYPASS",
  "_class": "HeaderVulnerability",
  "link": "https://www.cvedetails.com/cve/CVE-2018-8171/",
  "active": true
},{
  "_id": "33",
  "name": "ASP.NET Security Feature Bypass Vulnerability (Asp.net
↪  Core)",
  "versionRanges": [
    {
      "versionTo": "1.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    },
    {
      "versionTo": "1.1",
      "versionFrom": "",
      "comparisonType": "EQ"
    },
    {
      "versionTo": "2.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "X-AspNet-Version"
  ],
  "type": "BYPASS",
  "_class": "HeaderVulnerability",
  "link": "https://www.cvedetails.com/cve/CVE-2018-8171/",
  "active": true
},{
  "_id": "36",
```

```
  "name": "Joomla! - Unauthenticated SQLi",
  "versionRanges": [
    {
      "versionTo": "3.10.6",
      "versionFrom": "3.0.0",
      "comparisonType": "BET"
    },
    {
      "versionTo": "4.1.0",
      "versionFrom": "4.0.0",
      "comparisonType": "BET"
    }
  ],
  "text": [
    "Joomla"
  ],
  "type": "SQL_INJECTION",
  "_class": "MetatagVulnerability",
  "link": "https://developer.joomla.org/security-centre/874-20220305
↪ -core-inadequate-filtering-on-the-selected-ids.html",
  "active": true
},{
  "_id": "20",
  "name": "Spoofing: GLOBALELITETRADE LTD Company No. 13598051
↪ (Dissolved)",
  "text": [
    "globalelitetrade.com",
    "globalelietrade.com"
  ],
  "type": "IOC",
  "_class": "LinkVulnerability",
  "link": "https://db.aa419.org/fakebanksview.php?key=160178",
  "active": false
},{
  "_id": "26",
  "name": "HTTP Request Smuggling in Apache HTTP Server",
  "versionRanges": [
    {
      "versionTo": "2.4.52",
      "versionFrom": "2.4.0",
      "comparisonType": "BET"
    }
  ],
  "text": [
    "Apache"
  ],
```

```
  "type": "RCE",
  "_class": "HeaderVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2022-22720",
  "active": true
},{
  "_id": "35",
  "name": "vBulletin Remote Command Execution via widget",
  "versionRanges": [
    {
      "versionTo": "5.6.2",
      "versionFrom": "5.5.4",
      "comparisonType": "BET"
    }
  ],
  "text": [
    "vBulletin"
  ],
  "type": "RCE",
  "_class": "MetatagVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2020-17496",
  "active": true
},{
  "_id": "37",
  "name": "MediaWiki - MassEditRegex allows CSRF",
  "versionRanges": [
    {
      "versionTo": "1.35.5",
      "versionFrom": "",
      "comparisonType": "LT"
    },
    {
      "versionTo": "1.36.2",
      "versionFrom": "1.36.0",
      "comparisonType": "BET"
    },
    {
      "versionTo": "1.37.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "MediaWiki"
  ],
  "type": "CSRF",
  "_class": "MetatagVulnerability",
```

```
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2021-46147",
  "active": true
},{
  "_id": "38",
  "name": "Ghost - Allows all authenticated users to view admin API
↪  keys",
  "versionRanges": [
    {
      "versionTo": "4.9.4",
      "versionFrom": "4.0.0",
      "comparisonType": "BET"
    }
  ],
  "text": [
    "Ghost"
  ],
  "type": "PRIVILEGE_ESCALATION",
  "_class": "MetatagVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2021-39192",
  "active": true
},{
  "_id": "39",
  "name": "React DOM - Lack of escaping could lead to XSS",
  "versionRanges": [
    {
      "versionTo": "16.1.1",
      "versionFrom": "16.1.0",
      "comparisonType": "BET"
    },
    {
      "versionTo": "16.2.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    },
    {
      "versionTo": "16.3.2",
      "versionFrom": "16.3.0",
      "comparisonType": "BET"
    },
    {
      "versionTo": "16.4.1",
      "versionFrom": "16.4.0",
      "comparisonType": "BET"
    }
  ],
  "text": [
```

```
    "react-dom"
  ],
  "type": "XSS",
  "_class": "ScriptVulnerability",
  "link": "https://reactjs.org/blog/2018/08/01/react-v-16-4-2.html",
  "active": true
},{
  "_id": "40",
  "name": "RCE Error text",
  "text": [
    "Cannot execute a blank command"
  ],
  "type": "RCE",
  "_class": "KeywordVulnerability",
  "link": "https://www.exploit-db.com/ghdb/1098",
  "active": true
},{
  "_id": "41",
  "name": "PHP Shell/Backdoor - FierzaXploit",
  "text": [
    "FierzaXploit"
  ],
  "type": "IOC",
  "_class": "KeywordVulnerability",
  "link": "https://beneri.se/malware/?md5=02fbe02807a16d933f3436c0c4
↪  26ac6d",
  "active": true
},{
  "_id": "42",
  "name": "PHP Shell/Backdoor - Symlink Sa",
  "text": [
    "Symlink Sa"
  ],
  "type": "IOC",
  "_class": "KeywordVulnerability",
  "link": "https://github.com/JesseClarkND/AttackFiles/blob/master/S
↪  ymlink-Sa-v3.0.php",
  "active": true
},{
  "_id": "43",
  "name": "PHP Shell/Backdoor - Sole Sad",
  "text": [
    "Sole Sad &amp;amp; Invisible",
    "Sole Sad & Invisible"
  ],
  "type": "IOC",
```

```
  "_class": "KeywordVulnerability",
  "link": "https://beneri.se/malware/?md5=0eebffcade2a2a9076837f7425
↪  362205#generated-html",
  "active": true
},{
  "_id": "44",
  "name": "PHP Shell/Backdoor - TheAlmightyZeus",
  "text": [
    "TheAlmightyZeus"
  ],
  "type": "IOC",
  "_class": "KeywordVulnerability",
  "link": "https://beneri.se/malware/?md5=116eb77281ee5facb5286865bc
↪  1c3b7d",
  "active": true
},{
  "_id": "45",
  "name": "PHP Shell/Backdoor - RBBD",
  "text": [
    "RBBD Shell Backdoor"
  ],
  "type": "IOC",
  "_class": "KeywordVulnerability",
  "link": "https://beneri.se/malware/?md5=23d4fe0d19674f6c668355dd14
↪  45aee5",
  "active": true
},{
  "_id": "48",
  "name": "WordPress Core - Authenticated Code Execution",
  "versionRanges": [
    {
      "versionTo": "4.9.8",
      "versionFrom": "3.7",
      "comparisonType": "BET"
    },
    {
      "versionTo": "5.0.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "WordPress"
  ],
  "type": "RCE",
```

```
  "link":
↪  "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-8942",
  "_class": "MetatagVulnerability",
  "active": true
},{
  "_id": "52",
  "name": "WordPress Enfold Theme - Pagination verification",
  "text": [
    "avia-element-paging"
  ],
  "type": "XSS",
  "_class": "LinkVulnerability",
  "link": "https://www.exploit-db.com/exploits/50427",
  "active": false
},{
  "_id": "53",
  "name": "WordPress Enfold Theme - Multi Criteria",
  "text": [
    "MC enfold"
  ],
  "type": "XSS",
  "_class": "MultiCriteriaVulnerability",
  "link": "https://www.exploit-db.com/exploits/50427",
  "active": true,
  "firstVulnerabilityId": "22",
  "secondVulnerabilityId": "52",
  "secondVulnerabilityType": "LinkVulnerability"
},{
  "_id": "16",
  "name": "WordPress LearnPress Plugin",
  "versionRanges": [
    {
      "versionTo": "4.1.7.3.2",
      "versionFrom": "",
      "comparisonType": "LTEQ"
    }
  ],
  "text": [
    "LearnPress"
  ],
  "type": "SQL_INJECTION",
  "_class": "MetatagVulnerability",
  "link": "https://patchstack.com/database/vulnerability/learnpress/
↪  wordpress-learnpress-plugin-4-1-7-3-2-auth-sql-injection-sqli-vu
↪  lnerability",
  "active": false
```

```
},{
  "_id": "34",
  "name": "Log4j Indicators of Compromise",
  "text": [
    "log.exposedbotnets.ru",
    "abrahackbugs.xyz",
    "cuminside.club",
    "m3.wtf",
    "pwn.af",
    "rce.ee",
    "rs3c1.com",
    "x41.me",
    "300gsyn.it",
    "nazi.uy",
    "agent.apacheorg.xyz",
    "apacheorg.top",
    "300gsyn.it",
    "agent.apacheorg.top",
    "apacheorg.xyz",
    "teamtnt.red",
    "verble.rocks",
    "viperdns.xyz",
    "wdnmdnmsl.xyz",
    "cnc.mariokartayy.com",
    "decoding9og.000webhostapp.com",
    "dsrn.com.br",
    "fkd.derpcity.ru",
    "igot.verymad.net",
    "wegot.verymad.net",
    "bash.givemexyz.in",
    "bash.givemexyz.xyz",
    "agent.apacheorg.top",
    "agent.apacheorg.xyz",
    "bash.givemexyz.in",
    "automationyesterday.com",
    "ggdd.co.uk"
  ],
  "type": "IOC",
  "_class": "ScriptVulnerability",
  "link": "https://blogs.infoblox.com/cyber-threat-intelligence/cybe
↪  r-campaign-briefs/log4j-indicators-of-compromise-to-date/",
  "active": false
},{
  "_id": "29",
  "name": "Embedthis GoAhead 2.5.0 - arbitrary HTTP Host header",
  "versionRanges": [
```

```
    {
      "versionTo": "2.5.0",
      "versionFrom": "",
      "comparisonType": "EQ"
    }
  ],
  "text": [
    "GoAhead"
  ],
  "type": "CODE_INJECTION",
  "_class": "HeaderVulnerability",
  "link": "https://nvd.nist.gov/vuln/detail/CVE-2019-16645",
  "active": true
},{
  "_id": "46",
  "name": "File zipper (vulnerability) allows downloading
↪  server-side code",
  "text": [
    "Unzipper version"
  ],
  "type": "INFORMATION_LEAKAGE",
  "_class": "KeywordVulnerability",
  "link": "",
  "active": true
},{
  "_id": "47",
  "name": "PHP Info leakage",
  "text": [
    "This program makes use of the Zend Scripting Language Engine"
  ],
  "type": "INFORMATION_LEAKAGE",
  "_class": "KeywordVulnerability",
  "link": "",
  "active": true
},{
  "_id": "51",
  "name": "Leafmailer mail client used by hackers",
  "text": [
    "leafmailer.pw"
  ],
  "type": "IOC",
  "_class": "ScriptVulnerability",
  "link": "",
  "active": true
},{
  "_id": "49",
```

```
  "name": "VMWARE VSPHERE login portal",
  "text": [
    "VmrcPluginUtil.js"
  ],
  "type": "LOGIN_PORTAL",
  "_class": "ScriptVulnerability",
  "link": "",
  "active": true
},{
  "_id": "50",
  "name": "AppleJeus Cryptocurrency Malware",
  "text": [
    "celasllc.com",
    "jmttrading.org",
    "unioncrypto.vip",
    "kupaywallet.com",
    "CoinGoTrade.com",
    "dorusio.com",
    "ants2whale.com"
  ],
  "type": "IOC",
  "_class": "LinkVulnerability",
  "link": "https://www.cisa.gov/news-events/cybersecurity-advisories
↪  /aa21-048a",
  "active": true
}]
```

D

TLDs from scans

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned.

		Vulnerabilities		**Normalized vulnerabilities (min 1000 URLs/ 300 domains)**	
	Country	**Count URLs**	**Count domains**	**Vulnerable URLs**	**Vulnerable Domains**
.ac	**Ascension Island**	23 088	240	14,30%	18,75%
.ad	**Andorra**	10 267	91	8,89%	18,09%
.ae	**United Arab Emirates**	136 716	2 164	9,24%	15,18%
.af	**Afghanistan**	6 900	103	5,78%	12,89%
.ag	**Antigua and Barbuda**	8 412	277	5,36%	15,45%
.ai	**Anguilla**	47 640	1 396	5,88%	10,44%
.al	**Albania**	89 235	923	14,64%	21,53%
.am	**Armenia**	81 075	943	6,72%	15,46%
.ao	**Angola**	6 865	165	6,62%	22,73%
.ar	**Argentina**	1 392 644	19 164	13,78%	19,83%
.as	**American Samoa**	3 396	142	2,54%	8,07%
.at	**Austria**	1 698 380	34 571	11,29%	13,01%
.au	**Australia**	877 731	6 836	11,15%	14,07%
.aw	**Aruba**	159	11	2,33%	#N/A
.ax	**Åland Islands**	4 609	102	10,19%	12,42%
.az	**Azerbaijan**	37 961	677	2,97%	9,64%

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.ba	**Bosnia and Herzegovina**	135 725	1 439	11,25%	18,63%
.bb	**Barbados**	4 464	47	16,46%	#N/A
.bd	**Bangladesh**	91 622	975	12,97%	20,74%
.be	**Belgium**	1 744 951	30 391	9,48%	11,49%
.bf	**Burkina Faso**	5 190	91	7,64%	27,91%
.bg	**Bulgaria**	513 654	4 614	11,48%	19,95%
.bh	**Bahrain**	3 457	87	3,75%	19,91%
.bi	**Burundi**	4 420	53	5,83%	16,11%
.bj	**Benin**	3 277	93	5,41%	#N/A
.bm	**Bermuda**	7 353	83	14,96%	12,46%
.bn	**Brunei**	901	33	1,56%	#N/A
.bo	**Bolivia**	59 818	651	16,19%	31,15%
.br	**Brazil**	2 230 426	15 634	20,55%	33,83%
.bs	**Bahamas**	1 371	30	2,73%	#N/A
.bt	**Bhutan**	20 826	179	25,59%	43,55%
.bw	**Botswana**	4 297	117	4,92%	21,35%
.by	**Belarus**	349 378	4 012	6,66%	10,79%
.bz	**Belize**	18 166	298	7,29%	11,60%
.ca	**Canada**	3 638 847	41 192	11,75%	11,55%
.cc	**Cocos (Keeling) Islands**	145 380	2 421	4,24%	6,88%
.cd	**Democratic Republic of the Congo**	8 257	67	7,53%	17,14%
.cf	**Central African Republic**	4 946	282	2,61%	1,75%
.cg	**Congo**	565	17	0,97%	#N/A
.ch	**Switzerland**	1 749 565	31 566	8,68%	8,91%
.ci	**Ivory Coast**	14 314	194	8,88%	15,52%
.ck	**Cook Islands**	126	5	0,25%	#N/A
.cl	**Chile**	886 982	14 514	13,87%	18,84%
.cm	**Cameroon**	13 115	223	11,23%	16,21%
.cn	**China**	614 784	12 138	2,91%	3,91%
.co	**Colombia**	1 122 023	16 726	8,08%	10,40%

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.cr	**Costa Rica**	155 974	691	29,46%	32,63%
.cu	**Cuba**	84 734	245	17,42%	#N/A
.cv	**Cape Verde**	7 877	89	8,21%	23,00%
.cw	**Curaçao**	6 919	36	39,29%	#N/A
.cx	**Christmas Island**	35 086	138	36,58%	13,83%
.cy	**Cyprus**	44 448	574	7,45%	16,36%
.cz	**Czech Republic**	4 496 553	55 604	13,02%	16,89%
.de	**Germany**	12 734 344	211 783	9,15%	9,13%
.dj	**Djibouti**	2 898	33	3,99%	#N/A
.dk	**Denmark**	1 472 244	23 186	10,49%	11,17%
.dm	**Dominica**	236	17	0,45%	#N/A
.do	**Dominican Republic**	67 151	809	11,00%	19,84%
.dz	**Algeria**	42 217	610	15,17%	31,28%
.ec	**Ecuador**	173 178	1 775	16,56%	26,41%
.ee	**Estonia**	391 234	5 951	8,13%	14,63%
.eg	**Egypt**	51 403	491	13,27%	32,18%
.er	**Eritrea**	0	0	#N/A	#N/A
.es	**Spain**	4 307 414	49 086	13,57%	17,06%
.et	**Ethiopia**	41 447	164	37,68%	44,20%
.fi	**Finland**	1 683 313	22 582	13,82%	16,32%
.fj	**Fiji**	31 707	111	27,14%	28,39%
.fk	**Falkland Islands**	105	5	5,43%	#N/A
.fm	**Federated States of Micronesia**	95 146	469	4,53%	13,23%
.fo	**Faroe Islands**	8 470	147	6,55%	10,13%
.fr	**France**	6 929 521	105 118	11,22%	15,10%
.ga	**Gabon**	12 176	465	7,41%	5,33%
.gd	**Grenada**	1 134	43	5,29%	#N/A
.ge	**Georgia**	142 890	1 489	12,45%	15,43%
.gf	**French Guiana**	1 622	10	23,90%	#N/A
.gg	**Guernsey**	25 520	303	5,43%	7,78%
.gh	**Ghana**	12 012	209	7,94%	22,67%
.gi	**Gibraltar**	1 568	62	1,83%	14,87%

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.gl	**Greenland**	2 354	93	1,82%	13,32%
.gm	**Gambia**	1 813	39	3,02%	#N/A
.gn	**Guinea**	435	11	10,21%	#N/A
.gp	**Guadeloupe**	1 455	28	3,22%	#N/A
.gq	**Equatorial Guinea**	3 019	302	4,41%	5,34%
.gr	**Greece**	2 118 034	22 673	12,11%	17,83%
.gs	**South Georgia and the South Sandwich Islands**	3 048	51	6,08%	8,95%
.gt	**Guatemala**	36 804	638	11,59%	23,10%
.gu	**Guam**	779	5	#N/A	#N/A
.gw	**Guinea-Bissau**	495	2	18,11%	#N/A
.gy	**Guyana**	9 823	46	12,33%	13,45%
.hk	**Hong Kong**	536 526	5 625	15,50%	20,74%
.hm	**Heard Island and McDonald Islands**	1 034	8	6,53%	#N/A
.hn	**Honduras**	23 203	201	10,60%	25,97%
.hr	**Croatia**	633 102	7 037	13,41%	19,13%
.ht	**Haiti**	11 863	34	11,70%	11,26%
.hu	**Hungary**	2 419 091	29 182	13,74%	18,34%
.id	**Indonesia**	1 242 080	17 067	14,07%	26,98%
.ie	**Ireland**	628 800	8 760	10,01%	13,18%
.il	**Israel**	208 207	1 573	16,09%	27,22%
.im	**Isle of Man**	10 930	285	4,52%	10,34%
.in	**India**	2 060 716	30 118	11,42%	15,31%
.io	**British Indian Ocean Territory**	439 626	8 196	6,16%	10,13%
.iq	**Iraq**	8 343	148	4,55%	30,83%
.ir	**Iran**	889 887	12 876	8,55%	14,68%
.is	**Iceland**	179 209	2 087	9,99%	14,17%
.it	**Italy**	10 013 080	130 212	15,61%	19,37%
.je	**Jersey**	2 754	87	2,01%	10,06%
.jm	**Jamaica**	12 260	88	14,83%	25,58%
.jo	**Jordan**	15 039	180	7,04%	17,87%

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.jp	**Japan**	3 740 293	44 412	7,54%	10,85%
.ke	**Kenya**	155 453	2 268	14,74%	24,11%
.kg	**Kyrgyzstan**	53 238	587	9,36%	18,47%
.kh	**Cambodia**	19 430	221	11,71%	29,27%
.ki	**Kiribati**	231	5	0,58%	#N/A
.km	**Comoros**	90	5	#N/A	#N/A
.kn	**Saint Kitts and Nevis**	599	11	6,19%	#N/A
.kp	**North Korea**	557	2	6,50%	#N/A
.kr	**South Korea**	1 007 330	13 406	9,92%	10,85%
.kw	**Kuwait**	6 912	96	4,15%	14,20%
.ky	**Cayman Islands**	3 114	76	6,92%	9,82%
.kz	**Kazakhstan**	238 279	3 161	6,48%	11,59%
.la	**Laos**	24 576	361	6,90%	10,59%
.lb	**Lebanon**	12 614	165	6,30%	23,27%
.lc	**Saint Lucia**	3 989	30	24,37%	#N/A
.li	**Liechtenstein**	22 388	569	8,15%	12,56%
.lk	**Sri Lanka**	118 492	1 648	14,69%	25,86%
.lr	**Liberia**	295	11	7,05%	#N/A
.ls	**Lesotho**	1 452	51	2,40%	#N/A
.lt	**Lithuania**	633 256	10 827	12,05%	19,88%
.lu	**Luxembourg**	142 336	2 427	12,20%	17,26%
.lv	**Latvia**	379 668	4 733	12,22%	16,26%
.ly	**Libya**	17 194	334	5,28%	17,08%
.ma	**Morocco**	104 091	1 710	9,66%	18,25%
.mc	**Monaco**	5 142	137	7,66%	19,19%
.md	**Moldova**	123 823	1 337	9,35%	16,49%
.me	**Montenegro**	402 160	5 712	6,17%	8,43%
.mg	**Madagascar**	8 212	155	7,02%	23,17%
.mh	**Marshall Islands**	3	3	#N/A	#N/A
.mk	**Republic of North Macedonia**	167 838	1 677	14,44%	21,08%
.ml	**Mali**	8 760	234	4,94%	1,32%

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.mm	**Myanmar**	19 182	286	14,18%	36,76%
.mn	**Mongolia**	51 821	745	13,37%	22,95%
.mo	**Macau**	20 300	165	12,26%	25,04%
.mq	**Martinique**	3 298	30	22,40%	#N/A
.mr	**Mauritania**	1 918	33	9,79%	#N/A
.ms	**Montserrat**	6 183	109	4,84%	11,86%
.mt	**Malta**	48 450	399	12,87%	17,17%
.mu	**Mauritius**	17 643	268	8,34%	18,12%
.mv	**Maldives**	14 176	123	10,31%	19,68%
.mw	**Malawi**	4 227	81	5,55%	#N/A
.mx	**Mexico**	766 998	9 285	16,57%	25,13%
.my	**Malaysia**	505 934	6 437	14,92%	18,71%
.mz	**Mozambique**	8 835	265	6,75%	29,51%
.na	**Namibia**	23 118	121	13,97%	16,05%
.nc	**New Caledonia**	14 434	208	12,12%	14,85%
.ne	**Niger**	1 316	37	3,58%	#N/A
.nf	**Norfolk Island**	510	11	1,30%	#N/A
.ng	**Nigeria**	130 829	2 281	7,39%	18,39%
.ni	**Nicaragua**	16 905	210	11,19%	35,12%
.nl	**Netherlands**	3 924 164	94 007	7,14%	10,62%
.no	**Norway**	1 128 099	14 443	10,64%	11,87%
.np	**Nepal**	114 835	1 697	24,31%	23,15%
.nr	**Nauru**	0	0	0,00%	#N/A
.nu	**Niue**	204 961	3 544	11,41%	12,73%
.nz	**New Zealand**	258 343	2 679	11,92%	12,20%
.om	**Oman**	4 574	125	4,25%	18,49%
.pa	**Panama**	27 943	342	10,97%	25,37%
.pe	**Peru**	377 770	4 274	16,60%	25,85%
.pf	**French Polynesia**	7 878	62	16,10%	15,23%
.pg	**Papua New Guinea**	6 664	60	13,79%	17,14%
.ph	**Philippines**	203 263	3 623	11,42%	27,48%
.pk	**Pakistan**	205 030	2 792	9,14%	18,42%
.pl	**Poland**	5 111 645	83 064	9,87%	16,86%

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.pm	**Saint Pierre and Miquelon**	841	54	1,11%	11,79%
.pn	**Pitcairn Islands**	13	1	0,03%	#N/A
.pr	**Puerto Rico**	4 578	41	4,90%	#N/A
.ps	**Palestinian National Authority**	25 341	218	9,94%	16,80%
.pt	**Portugal**	1 140 547	12 328	12,49%	16,86%
.pw	**Palau**	9 428	431	2,62%	4,37%
.py	**Paraguay**	72 596	958	17,79%	25,47%
.qa	**Qatar**	16 756	238	7,43%	15,54%
.re	**Réunion**	24 773	704	10,09%	17,20%
.ro	**Romania**	1 653 777	23 115	9,47%	16,12%
.rs	**Serbia**	655 087	6 351	14,25%	18,15%
.ru	**Russia**	10 016 789	100 953	7,47%	11,31%
.rw	**Rwanda**	13 075	185	10,22%	25,98%
.sa	**Saudi Arabia**	57 876	1 132	7,99%	19,39%
.sb	**Solomon Islands**	6 675	15	11,01%	#N/A
.sc	**Seychelles**	8 501	90	6,53%	16,95%
.sd	**Sudan**	2 563	99	9,82%	24,75%
.se	**Sweden**	2 332 324	33 588	11,77%	13,35%
.sg	**Singapore**	190 524	4 926	6,99%	15,31%
.si	**Slovenia**	634 581	7 934	13,57%	18,29%
.sk	**Slovakia**	1 362 909	19 416	11,71%	17,69%
.sl	**Sierra Leone**	2 863	31	7,13%	#N/A
.sm	**San Marino**	5 373	67	4,41%	20,06%
.sn	**Senegal**	14 327	220	9,45%	22,92%
.so	**Somalia**	5 812	103	4,43%	5,53%
.sr	**Suriname**	1 968	68	3,74%	20,54%
.ss	**South Sudan**	0	0	#N/A	#N/A
.st	**São Tomé and Príncipe**	7 015	134	3,08%	9,12%
.sv	**El Salvador**	45 787	401	21,36%	35,36%
.sx	**Sint Maarten**	2 848	20	13,29%	#N/A

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.sy	**Syria**	9 261	61	8,36%	18,77%
.sz	**Eswatini**	994	36	10,67%	#N/A
.tc	**Turks and Caicos Islands**	1 919	51	5,72%	8,96%
.td	**Chad**	322	9	0,91%	#N/A
.tf	**French Southern and Antarctic Lands**	1 731	29	5,27%	#N/A
.tg	**Togo**	4 731	86	6,95%	28,48%
.th	**Thailand**	558 648	4 542	22,26%	47,53%
.tj	**Tajikistan**	26 672	283	13,97%	25,40%
.tk	**Tokelau**	20 828	356	7,14%	2,35%
.tl	**Timor-Leste**	5 013	360	4,34%	#N/A
.tm	**Turkmenistan**	7 974	120	9,03%	27,59%
.tn	**Tunisia**	64 999	961	12,99%	21,89%
.to	**Tonga**	37 250	703	4,29%	14,27%
.tr	**Turkey**	464 477	3 429	15,53%	24,78%
.tt	**Trinidad and Tobago**	10 330	104	10,38%	23,06%
.tv	**Tuvalu**	615 944	4 228	9,52%	11,48%
.tw	**Taiwan**	685 964	5 805	12,84%	27,85%
.tz	**Tanzania**	19 618	834	8,73%	27,74%
.ua	**Ukraine**	961 527	9 016	8,61%	17,93%
.ug	**Uganda**	39 095	425	12,97%	25,10%
.uk	**United Kingdom**	1 837 795	7 271	14,63%	15,13%
.us	**United States**	534 385	7 517	6,79%	9,20%
.uy	**Uruguay**	146 376	1 811	14,41%	19,27%
.uz	**Uzbekistan**	68 476	892	8,05%	13,48%
.va	**Vatican City**	330	2	0,36%	#N/A
.vc	**Saint Vincent and the Grenadines**	17 145	244	11,22%	9,37%
.ve	**Venezuela**	67 250	733	13,36%	26,34%
.vg	**British Virgin Islands**	2 601	25	3,10%	#N/A

Continued on next page

Table D.1: Full list of TLDs with their count and a vulnerability percentage of count normalized with total web pages/domains scanned. (Continued)

.vi	**United States Virgin Islands**	70	7	0,16%	#N/A
.vn	**Vietnam**	1 458 442	16 596	8,87%	15,53%
.vu	**Vanuatu**	1 137	48	2,11%	#N/A
.wf	**Wallis and Futuna**	207	8	1,20%	#N/A
.ws	**Samoa**	33 589	609	6,28%	2,08%
.ye	**Yemen**	1 709	14	9,42%	#N/A
.yt	**Mayotte**	1 105	39	8,01%	#N/A
.za	**South Africa**	231 824	1 123	22,38%	22,03%
.zm	**Zambia**	5 506	187	7,33%	36,17%
.zw	**Zimbabwe**	26 081	563	12,53%	21,73%