Bachelor thesis Computing Science

Radboud University

Discovery of information disclosure vulnerabilities in the software development process

Supervisor:
Erik Poll
erikpoll@cs.ru.nl

Author:
Maurice Dibbets
s1022543

Second reader:
Hugo Jonker
hugo.jonker@ou.nl

Internship supervisor:
Ralph Moonen
ralph.moonen@secura.com

April 2021

Abstract

Information disclosure vulnerabilities can expose weaknesses in the technologies that are used by organizations and can even lead to the leakage of personally identifiable information. This thesis is about information disclosure vulnerabilities that are introduced in the software development process of closed source web applications.

For the purposes of our research, an information disclosure vulnerability is defined as any issue that leads to the public disclosure of information that was not intended to be exposed to the public.

The prevalence of information disclosure vulnerabilities in the software development process is examined using the following five vulnerability types as a baseline:

I. Leakage of application source code in .git directories.

II. The disclosure of dependencies (i.e, 3rd party code that is used by an application) in dependency manager files (e.g, `Gemfile.lock` in Ruby on Rails and `package-lock.json` in Node.js) that might have publicly known security vulnerabilities associated with them.

III. Misconfigured cloud storage (Amazon Web Services, DigitalOcean, Microsoft Azure, Google Cloud Platform, Alibaba Object Storage, and Oracle Cloud storage).

IV. Unauthenticated access to databases (MongoDB, Redis or indirectly through Elasticsearch).

V. The disclosure of API keys in GitHub repositories.

Note that II) consists of two steps, namely, a) the gathering of dependency manager files from a list of web applications and b) checking for the existence of vulnerabilities in the dependencies.

Additionally note that I), II), and V) specifically target obtaining the source code and dependencies from a list of closed source web applications, whereas III) and IV) do not have to involve the source code of an application and could lead to the discovery of any type of leaked information.

For I, II) and III), we used subdomain enumeration to look for target applications and keywords that could be used to brute force bucket names. We also checked for the existence of lists of file names on the target applications.

For IV) we used Shodan to search for open ports and for V) we looked through GitHub repositories.

A methodology was constructed to gather, aggregate, and analyze the findings. The research into the disclosure of API keys in repositories and unauthenticated access to databases turned out to be infeasible and those vulnerabilities were not investigated further.

We only accessed the information that was needed to perform the research and we did not exploit the vulnerabilities that were identified. Furthermore, we notified the owners of the affected systems.

Finally, we provide a list of defensive strategies against information disclosure vulnerabilities and a GitHub repository containing the custom scripts used in this research, which can be found at [1].

The biggest difference between this thesis and prior work is that for II) we look at closed source applications instead of open source applications.

Contents

Acknowledgements

I would like to thank my supervisors Erik Poll, Hugo Jonker, and Ralph Moonen for the help that they provided when I was writing this thesis. The feedback that they have provided has been tremendously helpful to improve by academic writing, and the discussions that we had contributed many ideas that could be used to improve my research as well as a deeper insight into the research community and the importance of cooperating with others. I really appreciated how enthusiastic each of you were about this thesis and I would love to cooperate on more projects in the future.

1 Introduction

Modern web architectures consist of various components including databases, content delivery networks, and third-party dependencies. Version control systems are used to keep track of changes to the source code and interactions between different Web APIs have become more commonplace.

As a result of this complexity, information disclosure vulnerabilities (introduced in the software development process of closed source web applications) are likely on the rise.

In this thesis, the leakage of application source code (I), dependencies that are used by an application (II), files stored in cloud storage instances (III), data stored in databases (IV) and API keys (V) will be considered.

- (I) A noteworthy issue that can lead to information disclosure is the exposure of ".git" directories on web servers. In 2015, Internetwache.org found that 9700 of the Alexa Top 1M websites were leaking application source code [2]. This is a relatively small number of websites, but it is worth revisiting this issue to check how the situation has changed since then.
- (II) Some websites expose the fact that they are using vulnerable dependencies through dependency manager files such as `Gemfile.lock` in Ruby on Rails and `package-lock.json` in Node.js. An example where this of this can be found in a HackerOne report [3].
- (III) According to a recent news report [4], 93% of cloud storage deployments are misconfigured and while these misconfigurations are not solely limited to exposing stored data to everyone on the Internet, it is critical that such misconfigurations are caught quickly before they become the cause of a data breach.
- (IV) Publicly exposed unauthenticated databases have become a widespread issue and in 2020 a cybercriminal tried to ransom nearly 47% of all MongoDB databases that were exposed online [5].
- (V) In 2019 researchers from the North Carolina State University found that over 100,000 GitHub repositories were leaking API or cryptographic keys [6].

In the OWASP Top Ten 2017 [7], a list of commonly identified vulnerabilities in web applications as constructed by the application security (AppSec) community, the Open Web Application Security Project ranked the item "Using Components with Known Vulnerabilities" in ninth place (ranking is based on exploitability, prevalence, and impact). Furthermore, the item "Security Misconfiguration", which is what I) and IV) could be categorized under was ranked in sixth place.

In 2019, GitHub released a feature using Dependabot to check for vulnerable dependencies in JavaScript, Ruby, Rust, PHP, and Python applications [8].

In 2021, the number of supported languages has increased [9]. Hence, my hypothesis is that the main cause that out-of-date dependencies are still being used is because of backward-compatibility reasons where an organization has not yet had the time to modify their code to adjust for the changes in a dependency update.

That being said, according to GitHub 94% of dependency updates for Rails applications specifically, were nonbreaking. Hence, it is possible that the number of websites that

are found to be affected will be relatively low, but because research into this topic seems to be lacking and because this type of research can quickly become outdated it is still worth investigating this hypothesis because it could lead to surprising results.

This thesis discusses a methodology that could be used for the detection of information disclosure vulnerabilities and then puts this methodology into practice. The methodology involves the use of multiple free and open-source software (FOSS) applications that are able to discover each of the issues discussed before.

In short, we will attempt to identify:

I. *The leakage of .git directories* (Section 2.2). This was identified as a potentially interesting research area because of a blog post by Internetwache [2]. This vulnerability type can cause source code to be unintentionally disclosed.

II. *Outdated components that contain vulnerabilities* (Section 2.3). This choice was inspired by a number of submissions by bug hunters to bug bounty programs on HackerOne. This vulnerability type is also related to the source code of an application because it is caused by out-of-date dependencies with known vulnerabilities in them.

III. *Exposed cloud storage* (Section 2.4). We decided to look into exposed cloud storage instances because of the amount of press coverage that these issues get and the potential impact of such findings. The suggestion to research this area came from one of my supervisors Ralph Moonen, who has experience in the area of (in)secure cloud storage. This type of vulnerability can cause the disclosure of any type of information.

IV. *Unauthenticated access to databases* (Section 2.5). This was another suggestion from Ralph Moonen and again the potential impact of such findings was the reason to choose this as a vulnerability type to investigate further. The author of this thesis also saw a number of bug bounty submissions related to this topic and thought this was a good indication of the relevance of this vulnerability type. Like the previous vulnerability, this vulnerability can cause the disclosure of any type of information.

V. *API keys in GitHub repositories* (Section 2.6). This was the last suggestion that I got from a meeting with Ralph Moonen. Prior research into this topic from North Carolina State University [10] also piqued my interest because of the number of findings mentioned in the paper and so this was chosen as another research area. This vulnerability type is related to the source code of an application (and thus I. and to a lesser extent II. as well) because it occurs when a developer adds hardcoded API keys to the source code of an application hosted publicly on GitHub. It is less related to II) because API keys do not get leaked in dependency files, but I) can lead to the discovery of API keys in the source of an application that does not have a public GitHub repository.

I), II), and V) are each related to the source code of an application. III) and IV) can lead to more general information leakage, and could include the exposure of source code as well as many types of personally identifiable information.

Most prior work, which can be found in Section 2.3.2, looks for vulnerable dependencies in open-source applications instead of closed-source applications as is done in II). The other issues have already been analyzed in previous work and this thesis analyzes

how the situation surrounding information disclosure vulnerabilities has changed since that work was published (i.e, has the number of exposed systems increased or decreased).

Hence, the biggest difference between this thesis and previous work is in the methodology used for II), although this methodology is only slightly different from that used for open source applications (instead of having access to a source code repository containing dependency manager files, we now need to fetch them directly from a live website) and the fact that we only look at closed source applications in II) instead of open source applications.

Before looking for the vulnerabilities labelled as I), II), and III), we first need to perform subdomain enumeration as a preliminary step:

- For I), we need a list of subdomains that we can use to check for the existence of .git directories. This list of subdomains is first resolved using a DNS resolver to check which ones are active and then ports 80 and 443 are checked for web applications (for an overview of the entire subdomain enumeration process see Section 2.1.7 and Section 3.1.1).
- For II), we need a list of subdomains that we can use to check for the existence of dependency manager files. This list of subdomains is also first resolved using a DNS resolver to check which ones are active and ports 80 and 443 are checked for web applications.
- For III), we need a list of subdomains that can be used to guess the names used for cloud storage instances. In the final experiment, we did not end up using subdomains to derive our list of keywords because the second experiment (Section 3.2.4.2) showed that using organization names derived from domain names (e.g, www.google.nl is translated to google), provided us with better results. For more information about how we structured our three sets of experiments, see Section 2.

IV) and V) do not need the preliminary step of subdomain enumeration. For IV) we use Shodan and for V) we can look for organization names derived from domain names (e.g, www.security.nl is translated to security). We did not look for the information disclosure vulnerabilities described in Section 4.2 because we had to limit the scope of our research due to time constraints.

As stated in Section 2 this thesis does not contain an extensive list of all information disclosure vulnerabilities. Instead, it aims to explore a subset of these vulnerabilities, because of the fact that information disclosure is a large topic which requires us to limit the scope of our research due to time constraints.

1.1 Responsible disclosure

When performing security research, affected parties should be contacted to help them improve their security posture. Furthermore, no excessive measures should be taken when testing for the existence of a vulnerability (e.g, extracting entire databases, breaking systems, etc). Based on a recommendation from my supervisor at Secura, it was decided that all issues that were discovered in this research would be discussed with the Dutch Institute for Vulnerability Disclosure [11]. During the coordinated vulnerability disclosure process, we followed guidelines of the National Cyber Security Centre [12] and stuck to the general advice from the public prosecution service of the Netherlands (Openbaar Ministerie in Dutch) [13]. Obviously, [13] is not legal advice, but the information in this post was useful to get a quick overview of some of the issues concerning

our research. Section A.2 shows examples of the emails that we sent to organizations that were vulnerable to the information disclosure issues covered in this research.

1.2 Contributions

- A detailed research methodology that can be used to check for the five types of information disclosure vulnerabilities mentioned in this section. The research methodology also describes how to look for dependency vulnerabilities in closed-source applications that leak their lists of dependencies in dependency manager files.
- A comparison with the findings from Ján Masarik's master thesis (Masaryk University) [14], which this thesis draws much inspiration from.
- Suggestions to the InfoSec community that could be used to improve existing tools.
- Information about the prevalence of information disclosure vulnerabilities in Dutch organizations.
- The improvement of the national digital security posture of the Netherlands by disclosing our findings to affected organizations and by providing suggestions that can be used to protect against information disclosure vulnerabilities.
- A word list consisting of the top 5,000 most common Dutch subdomains.
- The improvement of automation processes at Secura B.V. by contributing tools and ideas.

1.3 Structure of thesis

Section 2 discusses the methodology that we used in each of our experiments. It gives a high-level overview and an explanation about the types of information disclosure vulnerabilities that we looked for.

Section 3 discusses the experiments that we performed. It describes the issues that we encountered during the applied parts of our research and shows each of the commands that were used. Readers who are only interested in the research findings can refer to Section 3.5.

Section 4 discusses how our research could be improved and a number of related research areas. It contains many examples of information disclosure vulnerabilities that we did not look for in this thesis.

Section 5 gives advice for system owners that could be used to protect against some types of information disclosure vulnerabilities. It is supposed to provide a guideline and it is not a replacement for a carefully thought out information security policy.

Section 6 contains the conclusions that we did and our reflections on the research process and the findings in this thesis.

Section A contains the appendices (figures and examples of responsible disclosure emails) that are referred to in the other chapters.

A GitHub repository containing the custom scripts used in this research can be found at [1].

2 Methodology

In this section, the steps taken to investigate the prevalence of the information disclosure vulnerabilities that are mentioned in the introduction are explained. The research methodology consists of several sections (sections 2.1, 2.2, 2.3, 2.4, 2.5, and 2.6) that cover how to set up a lab environment that allows you to explore research areas that are related to the topic of information disclosure vulnerabilities.

It is important to note that this thesis does not contain an extensive list of all information disclosure vulnerabilities. Instead, it aims to explore a subset of these vulnerabilities (the same subset that was mentioned in Section 1), mainly due to the fact that information disclosure is a large topic which requires us to limit the scope of our research due to time constraints. For some examples of topics that were left out of this thesis, see Section 4.

The first environment that is prepared in the research methodology is that which is needed for subdomain enumeration (Section 2.1). Subdomain enumeration is used to inform the research into .git directories (Section 2.2) and vulnerable third-party dependencies (Section 2.3). As discussed in Section 1, to look for .git directories and vulnerable third-party dependencies, we can use a list of subdomain URLs that are to be checked for file names.

Furthermore, the discovered subdomains could also be used to extract keywords that can be used in the misconfigured cloud storage (Section 2.4) section.

Each of these sections covers different types of information disclosure vulnerabilities and the type of information that is leaked (subdomains, databases, application source code, etc) also differs for each of them.

2.1 Subdomain enumeration

As a preliminary step for I), II) and III), we first need to perform subdomain enumeration. For an explanation of why this is needed we refer you to the end of Section 1, but briefly subdomain enumeration is required for the workflows described in Section 2.2, Section 2.3 and Section 2.4.

2.1.1 Explanation of the risk of disclosing subdomains

Subdomains are domains that are hosted on top of a root domain. For example, the domain www.example.com is a subdomain of example.com because www.example.com is lower in the DNS (Domain Name System) hierarchy. Subdomain enumeration is the process of discovering subdomains either by using passive methods or active methods. We discuss the difference between these methods in the next section.

The discovery of subdomains can be useful for an attacker because it allows them to identify the applications that are owned by a target organization and occasionally these could include internal applications or applications that have already been forgotten by the developer who set up the subdomain.

One attack that abuses this is a subdomain takeover, allowing an attacker to hijack a DNS record by creating an account on a third-party service, but this attack could also be accomplished when an attacker takes ownership of an IP block that used to be

owned by the target organization.

An example of a subdomain takeover scenario is that a developer could create a subdomain referencing an S3 bucket and remove the S3 bucket while forgetting to remove the DNS record for the subdomain. This would allow an attacker to take over the subdomain by creating an S3 bucket with the same name as the one that has now been removed, effectively replacing the S3 bucket with one that can contain malicious content. Such a scenario took place in a Magecart attack [15].

2.1.2 Background on the techniques for finding subdomains

A good resource on the many different techniques that are available for subdomain enumeration is [16] which discusses both passive (third party data sources are queried) as well as active (requests to the target domain and its DNS server are being used) techniques. Readers are advised to look at this book if they want to explore the topic of subdomain enumeration in depth.

Additionally, the Bug Hunter Methodology [17] also provides information on what approach can be used to find root domains to perform the subdomain discovery on, as well as a lot of other advice on how to perform information gathering on a research subject (e.g an organization, web application, etc).

2.1.3 Tools

The following tools were used to perform subdomain enumeration:

- *Amass* is, as stated on the GitHub page of the tool, an "In-depth Attack Surface Mapping and Asset Discovery" tool that can be used to probe multiple open-source data sources about the subdomains of a given root domain and also supports many other types of operations [18].
- *MassDNS* is a fast DNS resolver [19]. It can be used together with the output of subdomain enumeration tools to check what subdomains are valid.
- *Subfinder* is a "subdomain discovery tool that discovers valid subdomains for websites", as noted on its GitHub page [20]. It queries multiple data sources to do so, and unlike Amass it does not have a subdomain brute forcing option.
- *httpx* is a tool that can check for web applications on a specified range of ports [21].

2.1.4 Comparison of Subfinder data sources that require an API key

Because the free data sources already provided us with enough information to conduct our research (see Section 3.1.1) and to avoid unneeded costs, we stuck to the free data sources for all of the root domains.

This means that we ran into a couple of restrictions that caused us to have more extensive information about the subdomains of the top 10 domains than of the other domains. This could introduce some bias when we create a word list of the 5,000 most common words that are used for subdomains.

The restrictions were as follows:

- BinaryEdge allows for 250 free queries per month [22].
- Censys allows for 250 free queries per month [23].
- Cert Spotter allows for 100 queries per hour [24].
- Chaos does not seem to have many restrictions. It is invitation only at the moment, however.
- The DNSDB Community Edition allows for 500 queries per month [25].
- The GitHub API allows for 5,000 requests per hour [26] when using basic authentication or OAuth.
- Intelligence X allows for 10 requests [27].
- PassiveTotal allows for 10,000 monthly searches on a 30-day enterprise trial when a work email is used [28].
- Recon.Dev allows for 30 free API requests per month [29].
- Robtex has a free option, but it is rate-limited [30].
- SecurityTrails allows for 300 requests per month [31].
- Shodan allows for 10,000 free results per month on an academic plan [32].
- Urlscan.io allows for 1,000 requests per day [33].
- VirusTotal allows for 172,800 requests per month (this can be found on the API key page) [34].
- Spyse was excluded because it required payment or else was only restricted to a 5-day trial [35].
- ZoomEye allows for 10,000 results per month [36].
- ThreatBook was also excluded because it seemed to require payment as well and the signup process was not entirely clear to me.

Amass provides us with 55 data sources and Subfinder provides us with 32 data sources. Finally, we decided to use Subfinder because of an open issue in Amass (Section 3.1.1.1).

2.1.5 Word lists and configuration files

Subdomain enumeration tools (as listed in Section 2.1.3) brute-force subdomains based on a provided word list. Due to the large number of DNS queries that are sent in this process and the aggressiveness of this approach, we have decided not to explore this option.

The example config.ini file [37] from Amass was modified by adding new domains to the `scope.domains` section and by adding API keys for each of the supported data sources.

Additionally, the c99, Recon.dev, threatbook.cn, zetalytics.com, ZoomEye, and CIRCL.lu data sources were excluded because they are paid data sources and the free data sources already provide enough coverage for the purposes of our research.

During the research, a bug was discovered in Amass which motivated us to use Subfinder instead (Amass would crash too often when we were gathering subdomains). We describe how we modified the config.yaml file used by Subfinder in Section 3.1.1.3 and Section 3.2.1.1.

2.1.6 Environment setup

Below are the commands that were used to setup the lab environment. Note that Amass did not end up being used in the experiments, so it is possible to ignore the installation instructions for it when replicating the results of this thesis (see Section 3.1.1.1).

```
# full URL can be found by adding a + after the bit.ly URL
wget https://bit.ly/3qYMHky -O amass.zip
unzip amass.zip
mv amass_linux_amd64/ amass/
amass/amass enum -config config.ini
```

Listing 1: Amass setup

```
# full URL can be found by adding a + after the bit.ly URL
sudo apt update && sudo apt upgrade
sudo apt install golang-go
wget https://bit.ly/39D9oUZ
tar xfz subfinder_2.4.7_linux_amd64.tar.gz
```

Listing 2: Subfinder setup

2.1.7 Workflow diagram

Figure 2 on page 63 demonstrates how we do subdomain enumeration. Our subdomain enumeration workflow consists of three phases:

1. Querying data sources using Subfinder to obtain a list of subdomains (the data sources are listed in config.yaml). In the final experiment, we used every data source except for Spyse and ThreatBook (see Section 2.1.4).

2. Using MassDNS to obtain DNS records for each of the valid subdomains and use a bash script (cleanup.sh) to extract the domain names from the results.

3. Checking what domains are used to host web applications on port 80 or port 443 using httpx and outputting the resulting list of URLs to alive.txt.

As stated in Section 3.1.1.1, Amass was excluded in this step because of a bug that we encountered in the first experiment.

2.2 .git directories (I)

2.2.1 Explanation of .git directories and related information disclosure risks

Git uses .git directories internally to keep track of the commit history, local branches, hooks, and more [38]. When the source code of a website is managed in Git, cloning a Git repository containing the source code of an application to a directory that is used by a web server (so you can host your web app) could cause the contents of the .git directory to be leaked. As a result, an attacker can steal parts of the source code of an application by reading out the commit history from the now publicly available /.git/ directory and extracting the contents of commits. The attacker can also read commit messages which could be useful to learn about confidential projects of an organization

or to learn more about the developers that work on a team. Commit messages also contain email addresses that could be used for the social engineering of developers.

None of these issues are because of a vulnerability in Git, instead they are caused by human error. If a web server is misconfigured to allow access to the .git directory then it is not surprising that the contents of this directory would get leaked. Usually, you would not be able to simply directly access the source code used in the back end of an application because the web server is configured not to expose the files containing this source code.

To fix this issue within Git itself would require the entire design of Git to be modified and I can not think of a way that this issue could be avoided in the design of a version control system, but admittedly this might be because I have never developed my own version control system or because my understanding of version control systems is not yet complete enough to come up with a solution to this.

2.2.2 Background and related work

The main work on .git directories is the original paper by Internetwache.org that was cited earlier [2].

2.2.3 Tools

.git directories can be checked for using the *meg* tool, which is discussed in the "leakage of vulnerable third-party dependencies" section.

2.2.4 .git locations

The different locations that can be checked for are (from [39]):

- /.git/config
- /.git/HEAD
- /.git/logs/HEAD
- /.git/index

2.2.5 Environment setup

Meg is relatively easy to install and because it is written in Go it is also cross-platform.

```
sudo apt install golang
go get -u github.com/tomnomnom/meg
```

2.2.6 Workflow diagram

Figure 3 on page 64 reuses the subdomain enumeration workflow from the previous section and provides the output to meg. Meg reads in the list of URLs that were generated by httpx and checks for a list of paths (the ones listed in Section 2.2.4) to see whether any of the URLs expose .git directories.

2.3 Leakage of vulnerable third-party dependencies (II)

2.3.1 Explanation of vulnerable third-party dependencies and related information disclosure risks

Applications often use third-party modules or libraries to implement features because it allows developers to save time instead of reimplementing the same functionality multiple times.

When vulnerabilities are discovered in third-party modules or libraries, they should be updated to fix the security flaws. If developers forget to install an update, this could expose their application to security exploits. Hence, it is interesting to check what websites leak the list of dependencies that they are using in files that are available publicly, to check whether these dependencies contain any vulnerabilities or not. In this attack scenario, in contrast to Section 2.2, no source code is leaked.

A sampling bias could be present because the owners of websites that leak their list of dependencies might be less security-aware than the owners of other websites, meaning that the websites that are discovered to leak their list of dependencies are more likely to use insecure dependencies than websites that do not leak such a list. Indeed, in Section 3.3.3.6, we found that 809/811*100% = 99.75% of the domains that leaked their dependencies also used vulnerable dependencies.

2.3.2 Background and related work

Prior work includes [40] which discusses the identification of known vulnerable components at a smaller scale, [41] which discusses the attitude that developers have towards updating third-party dependencies, [42] which discusses how not keeping dependencies up to date does not always lead to exploitable vulnerabilities and [43] which discusses a lack of awareness of NPM package maintainers about the security issues that are involved with developing NPM packages.

The last two papers specifically focus on Node.js, demonstrating that more research is needed into the security of web applications built using Java, PHP, Python, Ruby, .NET, and others. The approach in this thesis does not use the information in these papers, but they are referenced here for the interested reader.

2.3.3 Tools

The following tools are used as a part of our workflow:

- meg [44] is a tool that is used to check for the existence of files when provided with a list of domains. The environment setup for meg is shown in Section 2.2.5.

The other tools are each used to check for known vulnerabilities in the discovered files:

- auditjs [45] is a tool that is used to identify known vulnerabilities in `package-lock.json` files.
- bundler-audit [46] is used to check for known vulnerabilities in `Gemfile.lock` files.
- OWASP Dependency-Check [47] is used to check for known vulnerabilities in a wide variety of dependency files, including `pom.xml` and `packages.config` files.

- Safety [48] is used to check for known vulnerabilities in `requirements.txt` files. It can also be used to check for issues in `Pipfile` and `Pipfile.lock` files if pipfile2req [49] is used to convert them to `requirements.txt` files.
- Local PHP Security Checker [50] is used to check for known vulnerabilities in `composer.lock` files.
- cargo-audit [51] is used to check for known vulnerabilities in `cargo.lock` files.
- nancy [52] is used to check for known vulnerabilities in `Gopkg.lock` files.

2.3.4 Dependency manager files

The dependency manager files that are checked for in this research are as follows (full paths are omitted). Some of the checks were inspired by the ones that are implemented in Dependabot:

(a) .NET
- packages.config

(b) Python
- requirements.txt
- Pipfile
- Pipfile.lock

(c) Java
- pom.xml
- build.gradle

(d) PHP
- composer.lock

(e) Ruby / Ruby on Rails
- Gemfile.lock

(f) Node.js
- package-lock.json

(g) Rust
- cargo.lock

(h) Golang
- Gopkg.lock
- Gopkg.toml

2.3.5 Environment setup and usage information

The instructions below can be used to setup the dependency checkers mentioned in Section 2.3.3 on a Debian based system. As also mentioned in that section, the setup instructions for meg can be found in Section 2.2.5. If the setup instructions listed below do not work in your own lab environment, it is also possible to reference the GitHub pages of each tool for up-to-date instructions.

.NET, Java

```
# full URL can be found in references
wget https://bit.ly/37h78Bt -O dependency-check-6.1.1-release.zip
unzip dependency-check-6.1.1-release.zip
# for Java Maven projects
$PWD/dependency-check/bin/dependency-check.sh --out . --scan pom.xml
# for NuGet packages
$PWD/dependency-check/bin/dependency-check.sh --out . --scan packages.config
```

Python

```
sudo pip3 install safety
sudo pip3 install pipfile-requirements
pipfile2req > requirements.txt
safety check -r requirements.txt > auditreport.txt
```

PHP

```
# full URL can be found in references
wget https://bit.ly/37ippyx -O local-php-security-checker
chmod +x local-php-security-checker
./local-php-security-checker --path=composer.lock > auditreport.txt
```

Ruby / Ruby on Rails

```
sudo apt install rails
sudo gem update
sudo gem install bundler-audit
bundler-audit check --update > auditreport.txt
```

Node.js

```
# Builtin npm audit command
sudo apt install nodejs
sudo apt install npm
npm audit > auditreport.txt
# auditjs (see https://www.npmjs.com/package/auditjs)
npx auditjs@latest ossi
```

Rust

```
sudo apt install cargo
cargo install cargo-audit
export PATH=$PATH:/home/$USER/.cargo/bin
cargo-audit audit > auditreport.txt
```

Golang

```
# full URL can be found in references
wget https://bit.ly/3uaNAc2 -O nancy
chmod +x nancy
# this only works when you also have a Gopkg.toml file
./nancy sleuth -p Gopkg.lock > auditreport.txt
```

2.3.6 Workflow diagram

Figure 4 on page 65 demonstrates how to discover vulnerable third-party dependencies. The "Other files" node represents the files from Section 2.3.4 that have not been included in the diagram. Each of these files are used as input for the dependency check tools, as shown in Section 2.3.5. Using the list of URLs generated in the subdomain enumeration workflow, meg is used to check for dependency manager files and the files are written to disk. Different dependency vulnerability checkers are then used to read in the discovered dependency manager files and each of them generates a vulnerability report that can be examined by the researchers.

2.4 Discovering misconfigured cloud storage (III)

2.4.1 Explanation of misconfigured cloud storage and related information disclosure risks

Misconfigured cloud storage instances such as open Amazon AWS S3 (Simple Storage Service) buckets or Azure Storage Blobs can leak confidential files to the public. By trying to resolve domain names that are associated with storage services (e.g, example.s3.amazonaws.com), files could be listed (if an anonymous user has ListBuckets permissions [53]) and extracted by an attacker. These files could include application source code, lists of customers, credit card details, or anything else.

Another technique to find misconfigured S3 buckets is to monitor certificate transparency logs for S3 buckets and to look for a list of permutations, as is done by the bucket-stream tool [54].

The former technique (i.e, the resolving of domain names) is explored in sections 3.1.4, 3.2.4, 3.3.4 and the latter technique (i.e, the monitoring of certificate transparency logs) is explored in 3.4.

The latter technique is more likely to discover S3 buckets that have been created by organizations outside of the Netherlands (because we do not use a list of keywords for it that is specifically targeted towards the top 25,000 domain names in the Netherlands whereas we do use such a list with the former technique) and is used to get a general sense of how common the problem of misconfigured S3 buckets still is at the time of this writing.

The findings in Section 3.4.5 should not be used to draw conclusions about the security of cloud instances configured by organizations in the Netherlands.

2.4.2 Background and related work

Prior work includes [55] which discusses how a researcher from UpGuard discovered a data leak in an organization that is responsible for many of the Fortune 100 companies, and [56] which discusses how a research team from vpnMentor discovered data about 30,000 individuals who did business with a cannabis retailer.

2.4.3 Tools

- *cloud_enum* can be used to check for public resources in AWS, Azure, and Google Cloud [57].
- *spaces-finder* can be used to look for publicly accessible DigitalOcean Spaces [58].

- *buckets.grayhatwarfare.com* is a software offering that can be used to search for AWS buckets and Azure Blobs through a web interface [59]. We did not use this tool in our research, but it was used by the Dutch Institute for Vulnerability Disclosure to investigate some of our findings related to S3 buckets (see Section 3.4).
- *bucket-stream* can be used to monitor certificate transparency logs for interesting Amazon S3 Buckets [54]. Those buckets could be attacked within hours [60].

2.4.4 List of keywords

The usage of *cloud_enum* and *spaces-finder* requires a list of keywords that are contained in the domains that are to be analyzed. We merged the fuzz.txt and SpacesNames.txt files from *cloud_enum* and *spaces-finder*, respectively, for this purpose, but during the experiments, we quickly discovered that the resulting word list was too long (see Section 3.1.4.1).

2.4.5 Environment setup

The instructions below can be used to setup the cloud_enum, spaces-finder and bucket-stream tools on a Debian based system.

```
# cloud_enum
git clone https://github.com/initstring/cloud_enum
cd cloud_enum
pip3 install -r requirements.txt
python3 cloud_enum.py -kf targets.txt -m mutations.txt -l results.txt
# spaces-finder
git clone https://github.com/appsecco/spaces-finder.git
cd spaces-finder
python3 spaces_finder.py -l hosts.txt -g mutations.txt -t 5 > results.txt
# bucket-stream
# Instructions can be found at https://github.com/eth0izzle/bucket-stream
```

2.4.6 Workflow diagram

Figure 5 on page 66 shows how to find misconfigured cloud storage instances. To look for misconfigured cloud storage instances, a list of keywords (which in our case is a list of organization names extracted from the domain name, e.g, www.example.com becomes example) and a list of mutations (based on the most common names for subdomains) is used and both lists are provided to the cloud_enum and spaces-finder tools, respectively.

We also used the bucket-stream tool [54] to perform frequent checks for exposed S3 buckets (including the ones that are located outside of the Netherlands), to get a sense of how often S3 buckets are being misconfigured on a wider scale. This is done by monitoring certificate transparency logs.

2.5 Discovering exposed databases (IV)

2.5.1 Explanation of exposed databases and related information disclosure risks

When databases are connected to the Internet without requiring authentication, anyone can run queries against that database. This means that any personal data that is stored in said database can be stolen by an attacker. Similarly to Section 2.4, the data stored in a database could be anything.

2.5.2 Background and related work

One report from October 2020 reported that an estimated 4.9 million databases were found to have been exposed to the Internet [61]. This number was based on search results from Shodan (see Section 2.5.3). Details on how many of these databases did not require any authentication were excluded from the report. Hence, it is possible that the number of databases that allow for unauthenticated access is much lower.

2.5.3 Tools

Shodan [62] is a search engine for the Internet. It allows researchers to look for publicly available devices and the ports that these devices expose. It also supports looking for vulnerabilities based on Common Vulnerabilities and Exposures (CVE) [63] along with other features.

2.5.4 Workflow diagram

Figure 6 on page 66 shows how Shodan can be used to query for exposed databases. Exposed databases can be discovered using the port filter in a Shodan query (e.g, the query `port:21` can be used to look for services running on port 21). Shodan is also able to find Internet of Things devices, printers, and other devices, but that is outside of the scope of this research.

2.6 API keys in GitHub repositories (V)

2.6.1 Explanation of API keys in GitHub repositories and related information disclosure risks

Developers often use API (Application Programming Interface) keys to interact with online services that are integrated into their applications. When an API key is used developers tend to hardcode the API key in their source code, but when this is done in a public GitHub repository, this can allow unauthorized access to the API. The impact of the leakage of an API key could be low (for example, in the case of Google Maps API keys) or high (for example, in the case of API keys that allow access to a GitHub account itself). The impact also depends on the permissions that are set for an API key, which is an aspect that we ignore in our research. We did not research API keys in GitHub repositories because of the issues discussed in Section 2.3.6.

2.6.2 Background and related work

There are many examples that could be pointed out here. Examples of bug reports which discuss this issue can be found at [64], [65] and [66]. The GitHub repository at [67] discusses the impact that leaking different types of API keys can have.

2.6.3 Tools

GitDorker [68] uses the GitHub search API to find repositories that contain potentially sensitive information.
git-all-secrets [69] allows for searching for secrets in public repositories / gists that are associated with a given target organization by making use of entropy.

2.6.4 Files

The usage of GitDorker requires the use of keywords to look for. Luckily, a list of keywords is already provided with the tool itself and can be found at [70].

2.6.5 Environment setup

The instructions below show how to setup GitDorker and git-all-secrets. Note that neither of these tools ended up being used in our experiments for reasons discussed in 2.6.6.

```
# GitDorker
git clone https://github.com/obheda12/GitDorker
python3 GitDorker.py -org target -t mytoken -d Dorks/alldorksv3 -e 5 -o findings.txt
# git-all-secrets
sudo apt install docker.io
sudo service docker start
docker run -it abhartiya/tools_gitallsecrets -token=mytoken -org=target
```

2.6.6 Workflow diagram

There are two approaches that we would like to consider to look for API keys in GitHub repositories:
The first approach uses GitDorker to send requests to the GitHub API that contain certain keywords that indicate that sensitive API keys are stored in the repository of an organization. This could work, but the rate-limiting of GitHub together with the risk of getting our GitHub account banned from the platform (see GitHub ToS at [71]) makes this approach risky.

The second approach uses Google BigQuery to search for API keys. Google BigQuery provides the option to search through GitHub using SQL queries [72]. For the on-demand pricing option, the first TB of query data per month is free, but for large-scale searches, we assume that this option could become expensive.

Because the amount of data returned for each query with the patterns that match API keys is unknown, I have decided to leave this topic out of scope for this thesis. The paper from North Carolina State University mentioned earlier also seems to mention the methodological problems that exist in this type of research: "Essentially, there is no existing tool that can be used to confidently mine GitHub at a large-scale." [10].

2.7 Directory structure of our lab environment

Because the commands that are used in Section 3 often reference file names or directory names that are specific to our lab environment, we show the directory structure here to avoid any confusion that this might cause:

```
/home/user/thesis/cloud
/home/user/thesis/cloud/cloud_enum
/home/user/thesis/cloud/spaces-finder
/home/user/thesis/cloud/bucket-stream
/home/user/thesis/cloud/alibaba-oss
/home/user/thesis/cloud/oracle-oss
/home/user/thesis/dns_resolution
/home/user/thesis/dns_resolution/massdns
/home/user/thesis/sampling
/home/user/thesis/sampling/dependencyfiles
/home/user/thesis/sampling/dependencyfiles2
/home/user/thesis/sampling/dependencyfiles3
/home/user/thesis/sampling/gitdirs
/home/user/thesis/sampling/gitdirs2
```

```
/home/user/thesis/sampling/gitdirs3
/home/user/thesis/subdomain_enumeration
/home/user/thesis/subdomain_enumeration/amass
/home/user/thesis/subdomain_enumeration/subfinder
/home/user/thesis/subdomain_enumeration/output
```

3 Experiments and evaluation

In this section, the workflows that have been set up in Section 2 are put into practice and the results are documented in a table. A list of the top 25,000 domains in the Netherlands was used for this investigation [73]. A GitHub repository containing the custom scripts used in this research can be found at [1].

Section 4 discusses the limitations of this set of experiments and future research areas. We did not explore the disclosure of API keys in GitHub repositories because of the reasons discussed in Section 2.6.6.

Three sets of experiments were performed with an increasing number of analyzed target domains. Namely, the ones covered in Section 3.1 (with 25 target domains), Section 3.2 (with 1,000 target domains), and Section 3.3 (with the full list of 25,000 target domains). The experiments were structured this way because it enabled us to identify any practical issues with the workflows early in the research and because it allowed us to exclude information disclosure vulnerabilities that yielded less interesting results than we initially expected from the remaining experiments (see Section 3.1.5). Thanks to this approach, it was discovered that there was an issue in the Amass tool or with the environment in which this tool was executed, which meant that we could switch to using the Subfinder tool instead (see Section 3.1.1.1).

Furthermore, because Subfinder crashed when gathering data about some of the domains in the second experiment, we were able to exclude those domains before starting the third experiment and it was possible to account for potential future crashes in the third experiment.

In Section 3.1 we looked for issues I) to IV) that were mentioned in Section 1.
In Section 3.2 we looked for issues I) to III) and we excluded IV) because of the large number of compromised databases we discovered (see Section 3.2 for a more detailed justification of this decision).
In Section 3.3 we also looked for issues I) to III).

3.1 25 target domains

As a first experiment, we examined the top 25 domains from the list of the top 25,000 domains in the Netherlands mentioned before. We performed subdomain enumeration and looked at vulnerabilities I)-IV) of the ones mentioned in Section 1.

3.1.1 Subdomain enumeration

In this section we put the workflow shown in Section 2.1.7 into practice. This section is a first attempt at establishing the subdomain enumeration process that will be further improved upon in Section 3.2.1 and Section 3.3.1.

3.1.1.1 Amass bug

In the first experiment, an issue was encountered while using Amass (IO Wait error) [74] causing us to switch to using Subfinder. Because we did not end up using Amass, we will not describe the steps that were used to configure it here.

3.1.1.2 Obtaining the top 25 websites

The top 25 websites from a custom-generated Tranco list of all .nl websites that was generated on 2021-02-19, were used to try out the subdomain enumeration process. For more information on how Tranco lists are generated, see the original paper [75]. To obtain the top 25 domains from the CSV file, the following commands were used:

```
wget https://tranco-list.eu/download/JVWY/1000000 -O domains.csv
cut -d ',' -f 2 domains.csv > domains.txt
head -n 25 domains.txt > top25.txt
```

3.1.1.3 Choosing data sources and finding subdomains using a passive approach

Once the top 25 domains were obtained, we modified the config.yaml file used by Subfinder to include API keys for Censys, Chaos, Intelligence X, RiskIQ PassiveTotal, SecurityTrails, Shodan, and VirusTotal. We excluded Robtex, Spyse, ZoomEye, and threatbook.cn in our initial experiment and we forgot to include BinaryEdge, Cert Spotter, DnsDB, GitHub, Recon.dev, and URLScan.

Nevertheless, these sources (together with the default sources of Subfinder that do not require API keys) resulted in 1,362,464 subdomains at the time of this writing and because more than a million subdomains were already obtained for the top 25 websites (from the top 25,000 that were to be analyzed), we decided to use a stratified sampling method to only extract a subset of the subdomains that were discovered. We get back to this point in Section 3.1.1.6.

The following Subfinder command was used to discover the subdomains:

```
./subfinder -nW -config /home/$USER/.config/subfinder/config.yaml -
    ↪ dL ../../top25.txt -rL ../resolvers.txt -oD ../output/ -t 50
```

The resolvers.txt file was obtained from [19] and the listed flags are described in the Subfinder help menu (`subfinder -h`). Note that the usage of nW is incorrect here as the output from Subfinder was not yet obtained, but we want to be as transparent as possible about the steps we took in our research.

3.1.1.4 Resolving passively discovered subdomains

Once the subdomains were obtained, it was still necessary to check which ones were resolvable. For this, the following command was used, which uses MassDNS to look up the A records of the list of subdomains:

```
cat ../../subdomain_enumeration/output/all.txt | ./bin/massdns -r
    ↪ ../../subdomain_enumeration/resolvers.txt -t A -q -w
    ↪ experiment.txt --filter NOERROR -o S
```

Once again, we notice possible improvements because the usage of the advanced flag "-m" in the simple output mode could be used to reduce the amount of output generated by MassDNS. The point of this toy example is to identify these types of improvements before we apply our tooling at a larger scale.

3.1.1.5 Cleaning MassDNS output

Now that we have a list of the returned DNS records for each subdomain, we can clean up the output of MassDNS to extract the list of active subdomains using the following bash script:

```
#!/bin/sh

# https://tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html
if [ ! $# == 2 ]; then
    echo "Usage: $0 massdns.txt output.txt"
    exit
fi

inputfile=$1
outputfile=$2

# https://unix.stackexchange.com
# /questions/76061/can-sed-remove-double-newline-characters
awk {'print $1'} $file | uniq | sed 's/;//g' | sed 's/\.$//' | sed '/^$/d' > $outputfile
```

Listing 3: cleanup.sh

3.1.1.6 Finding web applications

Finally, using httpx, we can check which of the resolved subdomains has a web application running on port 80 or port 443:

```
cat resolved.txt | httpx -title -threads 100 -o experiments.txt
```

This last step took multiple hours, due to the fact that a full TCP (Transmission Control Protocol) handshake had to be performed for each of the domains in resolved.txt and because our VPS had limited resources. For this reason, the following simple Python script was used to take the first 100 subdomains discovered for each root domain as a part of stratified sampling:

```
# Get 100 subdomains
# for every root domain

count_subdomains = {}

# https://stackoverflow.com/questions/9573244/
# how-to-check-if-the-string-is-empty
def is_not_blank(s):
    return bool(s and not s.isspace())

def is_blank(s):
    return not is_not_blank(s)

with open('../dns_resolution/massdns/domains.txt', 'r') as domains:
    with open('sampled.txt', 'w') as sampled:
        for subdomain in domains:
            try:
                sample = subdomain
                if is_blank(subdomain):
                    continue
                domain_components = subdomain.split('.')
                if len(domain_components) != 3:
                    continue
                domain_name = domain_components[-3]
                tld = domain_components[-1] # TLDs such as
                                            # .co.uk are not present
                                            # in our list
                root_domain = domain_components[-2] + '.' + tld
```

```
            if root_domain not in count_subdomains:
                count_subdomains[root_domain] = 1
                sampled.write(sample)
            else:
                if count_subdomains[root_domain] < 100:
                    count_subdomains[root_domain] += 1
                    sampled.write(sample)
    except (ValueError, IndexError) as e:
        print('Failed on ' + subdomain)
        print(e)
```

Listing 4: sample.py

3.1.1.7 Findings subdomain enumeration

After rerunning httpx, we could now use the resulting list of 850 subdomains (from the 5,998 subdomains that we obtained using the stratified sampling Python script) to look for .git directories.

3.1.2 .git directories (I)

Looking for .git directories became trivial and only took about 30 minutes thanks to the usage of meg. We renamed experiments.txt (the output of the httpx command used earlier in Section 3.1.1.6) to hosts and added the files listed in Section 2.2.4 to the paths to look for. Then we used the following command to find .git directories:

```
meg paths hosts gitdirs -t 15000 -s 200 -c 5 -H "User-Agent: Googlebot"
```

This basic approach to finding .git directories was also applied in Section 3.2.2 and Section 3.3.1.

3.1.2.1 Findings .git directories

Surprisingly, this already resulted in two vulnerable subdomains from the 850 subdomains that were looked through. The status flag from meg did not seem to work for us for an unknown reason, but by grepping for 200 in the index file that is saved by meg and then manually analyzing each of the results (still doable at a small scale), we were able to determine that grepping for "heads" or "refs" in each of the discovered files was a reasonable approach to look for leaked .git files:

```
grep '200 OK' gitdirs/index | awk {'print $1'} > interesting.txt
# https://stackoverflow.com/questions/11619500
# /how-to-cat-multiple-files-from-a-list-of-files-in-bash
cat interesting.txt | xargs cat
# Look for git repos
grep -r heads .
```

In the HTTP responses, a few HTTP/1.1 403 Ip Forbidden responses were identified. This suggests that some hosts put our VPS on a blocklist.

3.1.3 Leakage of vulnerable third-party dependencies (II)

To implement the next step, the paths file was modified to look for vulnerable third-party dependency files (the paths we used are listed in Section 2.3.4).

After the subdomain enumeration step was completed, many of the information disclosure vulnerabilities that can be detected using the presence of files became trivial to check for (this step took about 30 minutes). We grepped for 200 OK responses and

used the results to look for dependency files by looking at each of the matched files.

When performing this research at a larger scale, it is necessary to use keywords that often occur in dependency manager files instead of using this approach, where possible (e.g, requirements.txt files do not have easy keywords to grep for).

```
meg dependencies hosts gitdirs -t 15000 -s 200 -c 5 -H "User-Agent:
    ↪ Security research by maurice.dibbets at student.ru.nl"
# Modified headers to inform reviewers of log files that traffic was
    ↪  not of a
# malicious nature
```

We used the same approach again in Section 3.2.3, but in the final experiment (Section 3.3.3) we used a more sensible approach.

3.1.3.1 Findings vulnerable third-party dependencies

Unfortunately, no dependency manager files were identified. One possible reason for this is provided in Section 3.2.5 (we did not use full paths).

3.1.4 Discovering misconfigured cloud storage (III)

This section describes how we looked for misconfigured cloud storage. Because the findings in this section were disappointing (see Section 3.1.4.3) we changed the approaches in Section 3.2.4 and 3.3.4. The higher-level methodology that is used was described in Section 2.4.

3.1.4.1 Choosing keywords and using them to discover misconfigured cloud storage instances

To look for misconfigured cloud storage, it was necessary to find the most common keywords that were present in our list of (active/inactive) subdomains, so that those keywords could be combined with root domain names to find the largest number of misconfigured cloud storage instances. To do this, we used the full list of subdomains that was generated by Subfinder together with the following one-liner to get a sorted list of the 75 most common words used for subdomains:

```
# https://unix.stackexchange.com
# /questions/170043/sort-and-count-number-of-occurrence-of-lines
cat all.txt | cut -d '.' -f 1 | sort | uniq -c | sort -nr | awk {'
    ↪ print $2'} | head -n 75 > wordlist.txt
```

The root domain names were extracted using the `cut` command from the original list of domains as follows:

```
cat domains.txt | cut -d '.' -f 1 > roots.txt
```

Finally, these two files were used as a list of keywords and mutations for the cloud_enum tool in the following way:

```
python3 cloud_enum.py -kf ~/thesis/top25.txt -m ~/thesis/
    ↪ subdomain_enumeration/output/wordlist.txt -l cloudresults.txt
```

When we ran cloud_enum, we noticed that 11,275 mutated results were generated. cloud_enum checks 11 different services and in our research environment about 10 DNS queries were sent per second on average, hence it would take (11,275*11)/10 = 12,403 seconds or about 3 hours and 30 minutes to look for cloud instances for the top 25 domains alone.

Unfortunately, if we were to use 75 keywords for our full 25,000 domain list, it would take multiple months to complete. Because of this, we have decided to only use 14 keywords, which is far from ideal but could still yield some results.

3.1.4.2 Finding DigitalOcean spaces

Looking for DigitalOcean spaces using spaces-finder was faster than using cloud_enum because spaces-finder only needs to check one type of service (whereas cloud_enum checks 11 services). cloud_enum also does not support looking for DigitalOcean spaces yet, which is why we describe DigitalOcean spaces outside of Section 3.1.4.1, but DigitalOcean is no different from the other cloud providers we checked (at least in terms of our research methodology).

Interestingly, 60 DigitalOcean spaces were discovered using the following command:

```
python3 spaces_finder.py -l ../../roots.txt -d digitalocean -g interesting_keywords.txt
```

Most of these spaces only allowed for the listing of objects and did not allow us to actually download the objects. None of the spaces seemed to contain important information.

Later during the research in Section 3.2.4, I noticed that the command I used to check for DigitalOcean spaces already checks each of the 25,000 names from `roots.txt`. This makes the 60 DigitalOcean spaces somewhat less interesting.

3.1.4.3 Findings about misconfigured cloud storage instances

Every cloud instance that we discovered using cloud_enum was configured correctly, but by using spaces-finder we did discover 60 DigitalOcean spaces that provided us with LIST permissions. *spaces-finder* was already provided with a full list of 25,000 names from `roots.txt`.

3.1.5 Discovering exposed databases (IV)

We explored MongoDB, Redis, and Elasticsearch in this experiment. Elasticsearch is not a database, but it is often used to search through databases.

In the second and third set of experiments (see Section 3.2 and Section 3.3), we did not explore this area further because most of the databases we discovered had already been compromised and we did not think that reporting these compromises would do much to help the affected organizations (as mentioned in Section 3.1.6).

- *Discovering MongoDB servers.* There are three different ports that MongoDB can run on by default [76]: port 27017, 27018, and 27019. We only found MongoDB instances running on port 27017 and Shodan did not give any results when using the queries `port:27018` and `port:27019`. The queries for Shodan that can be used to look for unauthenticated MongoDB instances are already publicly documented [77] and most MongoDB instances (out of the 7,302 we discovered using Shodan, 135 of which were located in the Netherlands) seem to have been compromised and it takes a few seconds to find them, see Figure 1 on page 63.

  Because of this, not much is to gain from still reporting these instances to the affected organizations and this area will not be researched further. If anything, they are a good example of what can happen when you expose vulnerable systems to the Internet.

- *Discovering Redis servers.* Prior research has shown that 75% of exposed Redis servers have already been infected. This area will no longer be explored because of that fact [78]. It is notable that at the time of this writing, 13,887 Redis servers were found to be exposed to the Internet without requiring authentication. 308 of these instances were hosted in the Netherlands.
- *Discovering Elasticsearch servers.* Using a basic Shodan search, 16,221 Elasticsearch instances were found to be exposed to the Internet without requiring authentication. 333 of these instances were hosted in the Netherlands.

3.1.5.1 Findings exposed databases

To summarize the results from this section, we discovered $135+308+333 = 776$ exposed databases (meaning databases that did not require authentication) that are hosted in the Netherlands and $7,302 + 13,887 + 16,221 = 37,410$ of such databases in total.

3.1.6 Conclusion of experiment with 25 domains

Our initial experiments with just the top 25 domains seemed to suggest that misconfigured DigitalOcean spaces are more common than the other cloud storage options that we tested in Section 3.1.4, but in the second experiment (Section 3.2.4) when we looked back at the commands that we used in this section (Section 3.1.4), we discovered this was because of an error in the command that we used. The error was that we used roots.txt, the file containing all 25,000 domain names, instead of top25.txt, the file containing the top 25 domain names.

We also discovered that .git directories could be leaked more often than we initially expected, and we decided that the discovery of exposed databases was no longer interesting because most of the databases we found were already compromised so we would not be able to prevent much of the damage resulting from the misconfiguration by contacting the database administrators.

Finally, we found that the number of subdomains for popular domains can be quite large and we needed to limit our scope due to time constraints.

We are now left with four information disclosure vulnerabilities that are explored in the next experiments: subdomain enumeration, .git directories, vulnerable third-party dependencies, and misconfigured cloud storage.

3.2 1,000 target domains

In this section, we repeat the steps that we took when performing experiments on 25 target domains, but the commands are adjusted to remove flags that are not sensible (in Section 3.1.1.3, the nW flag was used incorrectly when running the Subfinder tool and the m flag of MassDNS was not used to reduce the amount of output that MassDNS generated) and the steps are performed on 1,000 target domains.

This section is intended as a last intermediate step before the full sample of 25,000 target domains is analyzed and some last major changes to our research methodology are made that are also described in this section.

As mentioned earlier, we only focus on subdomain enumeration, .git directories, vulnerable third-party dependencies, and the discovery of misconfigured cloud storage in

this second experiment. The reason we gave up on the discovery of exposed databases, as described in Section 3.1.5, is because a basic Shodan search informed us that the majority of exposed databases have already been compromised, meaning that our research cannot do much to help the affected organizations. Furthermore, the discovery of exposed databases seems to have been covered numerous times before and we decided it was best to focus our efforts elsewhere.

The second experiment is similar to the first one in Section 3.1, but some improvements were made to the commands that were used and, of course, the second experiment was done at a larger scale.

In the third experiment (Section 3.3), we do not list every command that we used (when no changes to the methodology are made) and list the results and provide tables of our final findings (the tables can be found in Section 3.5). Some exceptions are made because at the last stage of our research, we discovered there were a number of issues that we could look for that we did not include in the first two experiments, but that were easy to investigate (see Section 3.3.4.1 and 3.3.4.3).

We performed subdomain enumeration and looked at vulnerabilities I)-III) of the ones mentioned in Section 1.

3.2.1 Subdomain enumeration

This section improves upon the subdomain enumeration process that was used in Section 3.1.1. The steps below are the ones that will be followed in the final experiment (Section 3.3.1) as well. At the end of this section, we found 32,735 alive subdomains that we used in sections 3.2.2, 3.2.3, and 3.2.4.

3.2.1.1 Using Subfinder

In the second experiment, we used Subfinder and we applied it to the top 1,000 target domains:

```
head -n 1000 domains.txt > top1000.txt
```

Additionally, we did not use the nW flag and we did use the all flag to make the subdomain enumeration step more extensive (to include every data source except for Spyse and ThreatBook):

```
./subfinder -all -config /home/$USER/.config/subfinder/config.yaml -
    ↪ dL ../../top1000.txt -rL ../resolvers.txt -oD ../output/ -t 50
```

3.2.1.2 Subfinder crashes and problem resolution

Unfortunately, after a few hours, the Subfinder process crashed because it ran out of memory. The reason for this was probably that using all sources on a target domain that has dynamic IP addresses (e.g, 127-0-0-1.cable.dynamic.v4.ziggo.nl) causes too many results to be returned. In one of the attempts to mitigate this problem we ran the following command:

```
./subfinder -sources shodan -t 50 -config /home/$USER/.config/
    ↪ subfinder/config.yaml -dL ../../top1000.txt -rL ../resolvers.
    ↪ txt -t 50 -max-time 120 -silent -exclude-sources
    ↪ waybackarchive > ../output/ziggo.nl.txt
```

Notice that the top1000.txt file was never replaced and that the combination of -sources and -exclude-sources does not make sense.

We ended up with a file consisting of all the Shodan results for each of the top 1,000 domains that were to be investigated. We ran a few more commands that included only one data source to see whether it resolved the issue for the domain, but the process always crashed because it ran out of memory. Because of this, the coverage for the domain was not as complete as that of other domains.

3.2.1.3 Problematic domain names

We list every domain where we ran across this issue (including the ones we encountered in 3.3.1) to allow future researchers to focus on the more challenging attack surface:

```
% 2nd experiment
ziggo.nl
chello.nl
online.nl
versatel.nl

% 3rd experiment
direct-ads1.nl
orthovitality.nl
```

3.2.1.4 Resolving passively discovered subdomains and trying out new MassDNS output mode

The resulting subdomains were resolved using MassDNS:

```
cat ../../subdomain_enumeration/output/all.txt | ./bin/massdns -r
    ↪ ../../subdomain_enumeration/resolvers.txt -t A -q -w resolved.
    ↪ txt --filter NOERROR -o S
```

The process hung when using `-o Sm` so we decided to use the simple output mode again. The output was cleaned up using `cleanup.sh` (see listing 3) and sampled using `sample.py` (see listing 4).

3.2.1.5 Findings subdomain enumeration

After running httpx, we obtained a list of 32,735 alive subdomains that we used in sections 3.2.2, 3.2.3, and 3.2.4.

3.2.2 Findings .git directories (I)

We did not change the approach used to look for .git directories compared to the prior experiment with 25 target domains (Section 3.1.2).

Finally, we found that 27/1000*100 = 2.7% of the organizations that we tested were leaking application source code. By visiting each of the 27 corresponding domains we were able to verify that none of the applications that leaked source code were supposed to be open source.

Conclusion

The main industry that seems to be affected by this vulnerability is charities/non-profits (7 out of the 30 identified vulnerable subdomains), but the other subdomains include those owned by museums, government organizations, news organizations, book publishers, labor unions, and more.

3.2.3 Findings vulnerable third-party dependencies (II)

No vulnerable third-party dependencies were identified. There are a couple of reasons why this might be the case (the first was also mentioned in Section 3.1.3).

First, in our usage of meg, we did not add full paths to the dependency manager files (e.g, vendor/composer.lock). In our final experiment, this should be done. We also used a tedious process of manually reviewing responses because we did not spend time on specifying what pattern each of the types of dependency files follows (creating regular expressions for each type of dependency file could be tricky for some of the file formats and at a small scale it seemed easier to exclude 200 OK responses that included HTML tags by repeatedly using the grep utility).

Second, the usage of model view controller (MVC) frameworks by many web applications means that dependency files and directories such as `/vendor` are usually not exposed unless an application is misconfigured. Hence, it is likely that this attack vector is either uncommon or that our research methodology caused us to miss multiple instances of this vulnerability.

3.2.4 Discovering misconfigured cloud storage (III)

As mentioned in Section 3.1.4, we will change the way we look for misconfigured cloud storage in this section. In Section 3.3.4 the approach listed in this section will be used.

3.2.4.1 Generating updated word list with higher quality entries

A new word list was generated based on the updated list of all subdomains gathered by the passive enumeration tools. By manually examining the word list consisting of 15 items, it was noticed that a lot of the entries were based on domains that use dynamic IP addresses. Because of this, a higher quality word list was generated based on the alive subdomains discovered through httpx in Section 3.2.1.5, using the following command:

```
# https://askubuntu.com
# /questions/1087885/why-did-the-command-uniq-c-put-a-whitespace-at-
    ↪ the-beginning
cat ../../sampling/alive2.txt | cut -d '.' -f 1 | sort | uniq -c |
    ↪ sort -nr | awk {'print $2'} | sed 's/https:\/\///g' | sed 's/
    ↪ http:\/\///g' | head -n 15 > wordlist.txt
```

The word list was then used with the cloud_enum tool in the same way as was done in the experiment with 25 target domains.

3.2.4.2 Finding the optimal research strategy

Because the results from the previous experiment (see Section 3.1.4.3) were disappointing, I also used the first 1,000 lines of the roots.txt file (consisting of company names such as google) instead of only the top1000.txt file (consisting of the top 1,000 domain names such as google.nl).

In an attempt to find the research methodology that resulted in the most findings, I used the following three strategies:

1. Use the top 1,000 domain names of the first word list that was generated based on all subdomains gathered by passive enumeration tools (including unresolvable ones). By reading through the log file generated by the cloud_enum tool, I noticed that the only result was a Google Cloud storage instance that was supposed to be public because it hosted non-sensitive static content.

2. Use the first 1,000 lines of the roots.txt file together with a word list generated from resolvable subdomains gathered by passive enumeration tools. This resulted in three interesting Firebase instances [79] (one consisting of date of birth, school, and name information, another consisting of sales information, and yet another containing a command injection payload and looking like it might have been compromised) and an interesting misconfigured Google Cloud instance containing plans for the construction of a building.

3. Use the first 1,000 lines of the roots.txt file together with the 14 keywords that I came up with. The 14 keywords I used are:

```
dev
development
test
testing
stage
stg
int
internal
corp
corporate
backup
private
prod
production
```

   This resulted in the same three Firebase instances and misconfigured Google Cloud instance as the previous strategy, but also five additional Firebase results that seemed interesting. One of the issues I kept encountering is that I do not know where the data I am looking at comes from and so I cannot be sure whether it is meant to be public or not. Two of the additional Firebase results had been compromised, one of which contained a mapping between full names and usernames and contained group information.

   Another Firebase result contained shipping information for an unknown organization. There was also a restaurant leaking credentials of a tipping machine, but unfortunately, as with previous results, it was not entirely clear what restaurant, making it hard to do coordinated disclosure. Finally, there was also a social media website that was leaking information.

   Overall, the results of the experiment were not entirely clear and it was difficult to find out what organizations leaked the information that I discovered.

3.2.4.3 Findings misconfigured cloud storage

By trying out the three strategies listed in Section 3.2.4.2, we discovered 8 misconfigured Firebase instances and one misconfigured Google Cloud instance.

3.2.4.4 Best strategy and conclusion about misconfigured cloud storage

The third strategy was the most effective and I was able to confirm that three of the top 1,000 organizations I tested were leaking internal information and one of them had been compromised.

For the final experiment, only the third strategy is used. I tried to determine how sensitive the disclosed information that I discovered was based on how easy it would

be for a criminal to abuse the information that I discovered, but as this depends on the goals of said criminal and on the way in which an organization operates (e.g, some organizations consider an email address to be sensitive, but others do not), it is possible that I either overestimated or underestimated how many cloud instances leak sensitive information.

3.2.5 Conclusion of experiment with 1,000 domains

A significant number of organizations (27 out of the top 1,000 Dutch domain names) are still leaking source code through .git directories. The original research by Internetwache [2] found that less than 1% of the Alexa Top 1M was vulnerable to this attack, but when including subdomains, the percentage seems to be higher than that.

The search for third-party dependencies did not lead to interesting results and the approach to look for them was amended one last time in the final experiment by including full paths to dependency manager files instead of using a naive approach of looking for file names.

The results from our research into misconfigured cloud storage instances seem to indicate that Google Cloud Storage and Google Firebase instances are more commonly incorrectly configured than the ones from other cloud providers. Note that this might be because misconfigured S3 buckets get more press coverage than other types of misconfigured storage instances, meaning that cloud engineers might be more aware of the risks of misconfiguring them.

3.3 25,000 target domains

In the final experiment, the steps from Section 3.2 are repeated on a sample of 25,000 target domains and the results are collected in the subsections below. In Section 3.5, we compare the results from this experiment with the results from prior work and we evaluate our findings. The experiments in this section took the longest to execute and show that the approach presented in this thesis does not scale well when researching tens of thousands of domains. In the end we scaled down our research in many of the subsections below and we mention the real number of target domains that we investigated, which was often lower than the 25,000 target domains that we originally wanted to look at.

We performed subdomain enumeration and looked at vulnerabilities I)-III) of the ones mentioned in Section 1.

3.3.1 Findings subdomain enumeration

The process that was followed is largely the same as the one described in Section 3.2.1 with some exceptions:
For the first 5,860 domains, all data sources were used. Due to time constraints, only the default sources of Subfinder were included for the remaining 19,140 domains (the default sources can be found in the sources section of the `config.yaml` file created by Subfinder in the `/home/$USER/.config/subfinder/` directory). The Subfinder documentation for the all flag also states that using all sources is slow: `Use all sources (slow) for enumeration`.

In total, 12,141,105 subdomains were discovered using passive data sources. 6,403,917 (around 53%) of these subdomains were resolvable and 308,141 of the resolvable subdo-

mains were sampled. Of these sampled domains 252,163 (around 82%) were found to have web applications running on default ports and these subdomains are used in the next two sections (Section 3.3.2 and Section 3.3.3). In Section 3.3.4 we did not need the results of the subdomain enumeration because we used the optimal strategy that we discovered for finding misconfigured cloud instances in Section 3.2.4.4.

3.3.2 Findings .git directories (I)

Using the approach outlined in Section 3.1.2, we were able to discover 349 unique vulnerable domains (and 789 vulnerable subdomains) that leaked source code through .git directories. As before, the types of industries that were affected by this vulnerability varied widely, but non-profits were still the most commonly identified industry. Around 1.4% of the Dutch organizations (NOT considering the fact that some organizations might own multiple domain names) with the top 25,000 domain names leak the source code of one or more of their applications in .git directories.

3.3.3 Leakage of vulnerable third-party dependencies (II)

In this section (based on the results from Section 3.1.3.1 and 3.2.3) we adapt the approach that is used to look for vulnerable third-party dependencies one more time, and this time we were able to obtain some interesting results (see Section 3.3.3.4).

3.3.3.1 VPS storage issues

During the first attempt to collect third-party dependencies, my VPS ran out of storage space because of the size of some of the HTTP responses that were returned.

Thankfully, by revisiting the GitHub page of the meg tool [44], I discovered that there was an undocumented no-headers flag which could allow for much less storage to be used. This was also beneficial because it would no longer be needed to strip HTTP response headers from the responses that were received so that I could get valid dependency files to provide to the dependency vulnerability checking tools.

3.3.3.2 Meg crashes

Because of reasons that became unclear to me in my research, meg often seemed to crash while it was gathering results in a similar way that Amass did in Section 3.1.1.1 with an IO Wait error. This was probably because of network timeouts. On a more positive note, the status flag seemed to work in this experiment, unlike what happened in Section 3.1.2.

3.3.3.3 Using manual inspection to construct a list of keywords

Despite the issues with meg, we still managed to look at 13,567 domain names. This is far short of the 25,000 domain names that we initially wanted to look at, but it can still give an idea about the prevalence of this issue. To detect the dependency files, we used a list of 85 keywords (see Figure 7 on page 67 or use `excluded.txt` from [1], which contains a shortened version) to exclude from gathered results which were based on a manual inspection of the responses that were stored by meg (we redacted the IP address of the VPS that we used by replacing it with VPS_IP_ADDRESS and we replaced an organization name by COMPANY_NAME).

3.3.3.4 Findings about the number of domains leaking dependency files

In this case, we actually did find some dependency files that were being leaked unlike in the previous two experiments. Part of the reason for this is that we included full paths

in our usage of meg in this experiment and because we found a better way to filter out uninteresting responses (using grep together with the recursive `r`, case insensitive `i`, files without match `L`, fixed strings `F` and file `f` flags).

At the end of the experiment we found that 564 domain names seemed to be affected, but in Section 3.3.3.5 we actually found 811 domains that leaked dependency files (showing that the approach of excluding keywords does not work well). As stated before, we did not check the full 25,000 domain names and because we only saved responses with a status code of 200, it was not possible to figure out the exact number of domains that we tested retrospectively. However, based on the last response that was saved, at least 13,567 domains were tested and we know that around 811/13,567*100% = 5.98% of those domains leaked their dependency files. The reason we use the word "around" here is because even with all of the keywords above, there were still a number of responses that did not actually contain dependency files that had to be manually filtered out.

3.3.3.5 Further manual inspection

Because I manually reviewed the responses stored by meg, I noticed that most dependency files that were generated by the Composer tool had the word "getcomposer" in it, and by grepping for this word I actually found 811 domains that leaked `composer.lock` files.

Furthermore, I also found one requirements.txt file. The leakage of dependency files is not a vulnerability by itself, but it can provide an attacker with information about whether outdated dependencies are used by a website. The command I used was:

```
grep -r "getcomposer.org" . | awk {'print $1'} | sed 's/://g' > composerlockfiles.txt
```

3.3.3.6 Findings about the number of domains using vulnerable dependencies

By using the tools from Section 2.3.3, I was able to find 809 domains that used dependencies with known vulnerabilities (out of 13,567 domains, meaning that 809/13,567*100% = 5.96% of the domains were affected). Of these domains, 808 leaked composer.lock files and one leaked the requirements.txt mentioned earlier. This means that a surprising 99.63% of the domains leaking `composer.lock` files were vulnerable and 100% of the domains that leaked `requirements.txt` files (but there was only one domain that did so).

A bit of nuance is required here because the existence of an outdated component does not necessarily mean that it is used in a vulnerable way. Nevertheless, these numbers do seem to imply that either there is a serious bug in the `local-php-security-checker` tool that was used to check the `composer.lock` files or that the leaking of dependency files is highly indicative of vulnerable components being used on a domain. As mentioned in Section 2.3.1, however, this is probably because of a sampling bias.

Another surprising result is that I only discovered `composer.lock` and `requirements.txt` files, especially given the long list of files that we looked for across subdomains (provided in Section 2.3.4). It seems likely that some of these dependency files might have been excluded unintentionally because of the large list of keywords that we used to ignore responses.

3.3.3.7 Validity of findings

Because we only started finding dependency files in this last experiment and this ex-

periment had to be finished quickly due to time constraints, it would be interesting to see future research that explores the validity of our findings and any mistakes that we might have made.

3.3.4 Discovering misconfigured cloud storage (III)

In this section, we use the approach from Section 3.2.4 to look for misconfigured cloud storage instances, and we also looked at Alibaba Object Storage and Oracle Cloud instances. Because of the addition of these two new instance types, this section is a bit longer than Section 3.1.4 and Section 3.2.4.

3.3.4.1 Finding misconfigured Alibaba Object Storage instances

[14] described a workflow that can be used to look for misconfigured Alibaba Object Storage Service domains but did not include the results of performing this workflow. Hence, I decided to look for Alibaba Object Storage Service domains by using the following Python script and applying httpx to the resulting text file:

```
KEYWORDS = ["dev", "backup", "develop", "int", "internal", "staging", "test"]
with open("../../roots.txt") as roots:
    with open("targets.txt", "w+") as targets:
        for domain in roots:
            for keyword in KEYWORDS:
                target = domain.strip("\n") + "-" + keyword.strip("\n") +
                    ".oss.eu-west-1.aliyuncs.com" + "\n"
                targets.write(target)
```

Listing 5: generate_targets.py

The httpx command that I used was:

```
httpx -l targets.txt -match-string nosuchkey -o vuln.txt
```

Note that the -match-string flag (which is case insensitive) was used with "nosuchkey" because according to the paper mentioned earlier, if the code returned in a GET request to one of the *.oss-eu-west-1.aliyuncs.com domains is "NoSuchKey" then it is possible that we have read access to that domain.

3.3.4.2 Findings Alibaba Object Storage instances

Unfortunately, we did not find a single misconfigured Alibaba Object Storage Service (OSS) instance using this method. There are a couple of reasons that this might be the case:

- The first reason is that none of the Dutch organizations that we tested has a misconfigured Alibaba OSS domain. This is a possibility because Amazon S3, Azure, and Google Cloud are the most popular cloud service providers according to a report by Gartner from 2019 [80]. Note that this could have changed in 2021.
- The second reason is that the workflow that was used specifically looks for domains in the oss-eu-west-1 region. Perhaps some of the organizations that we examined use a different region, but note that the Netherlands is indeed a Western European country meaning it is likely that the storage locations would be chosen in this region.
- Other possible reasons include that the list of keywords that we used was too short, that there is a bug in one of the open-source tools that we used or that we overlooked an error in our approach. Because of the lack of public information on the prevalence of misconfigured Alibaba OSS instances, it is difficult to compare the results from this research with known data.

3.3.4.3 Findings misconfigured Oracle Cloud instances

We also briefly looked at misconfigured Oracle Cloud instances by following the workflow from the paper [14], but no vulnerable instances were identified by using the commands below:

```
mkdir oracle-oss
cd oracle-oss
cp ../../roots.txt .
sed 's/^/https:\/\//g' roots.txt > rootshttps.txt
sed 's/$/\.compat\.objectstorage\.eu-amsterdam-1\.oraclecloud\.com
    ↪ \//g' rootshttps.txt > targets.txt
# https://geek-university.com/linux/merge-files-line-by-line/
paste -d '' targets.txt roots.txt > targets2.txt
sed 's/$/\/test\//g' targets2.txt > targetsfinal.txt
httpx -l targetsfinal.txt -match-string nosuchkey -o vuln.txt
# Just to be sure match-string is really case insensitive,
# this command is redundant.
httpx -l targetsfinal.txt -match-string NoSuchKey -o vuln.txt
```

Note that in this case we did not use any keywords to look for misconfigured instances and only looked for domains following the pattern of:
`organizationname.compat.objectstorage.eu-amsterdam-1.oraclecloud.com`
with a bucket name being equal to the organization name so the final requests looked like:
`http://organizationname.compat.objectstorage.eu-amsterdam-1.oraclecloud.com/organizationname/`

3.3.4.4 Issues with cloud_enum tool and restricting the scope of our research

Finally, we also used the cloud_enum tool in the same way as we did in Section 3.2.4 and we used the third strategy for selecting keywords that was mentioned in that section. A bug was encountered which caused the tool to crash when brute forcing Azure Blobs. At this point in the research, we had already gathered so many vulnerable domains that had to be disclosed in the responsible disclosure process, that we decided against reviewing each of the cloud_enum results, again because the disclosure of the already discovered issues was considered of a higher priority.

3.3.5 Conclusion of experiment with 25,000 domains

This section posed numerous challenges because of the fact that 25 times more domains were targeted than in Section 3.2 causing us to run into time constraints. Nevertheless by making some changes to our approach (which ideally would not have been necessary after the previous section), we were still able to complete each of the subsections and gather our findings.

In Section 3.3.2 we found that 1.4% of the top 25,000 domain names in the Netherlands leak the source code of one or more subdomains in .git directories.

In Section 3.3.3 we were able to identify a number of websites that leaked the dependencies that they were using. The findings were limited to dependency files that are associated with Python and PHP applications and 809/13,567 = 5.96% of the tested root domains leaking their dependencies used dependencies with known vulnerabilities in them. We did not look at the impact of these vulnerabilities.

In Section 3.3.4 we did not find any vulnerable Alibaba Object Storage Service or Oracle Cloud instances, but as discussed in that section, this could be because of errors

in the methodology that we used, because of bugs in the open-source tools and because of various other reasons. As was also discussed in that section, we decided against reviewing each of the discovered cloud storage instances because we wanted to prioritize the reporting of the bugs that had already been discovered (these reports had to be sent to more than 300 organizations).

3.4 Live monitoring of buckets (III)

To get a more general sense of how common the issue of misconfigured S3 buckets is, we used the bucket-stream tool to perform monitoring of certificate transparency logs for a list of default permutations that can be found at [81]. As mentioned in Section 3.4, this section is not supposed to be used to draw conclusions about the prevalence of this vulnerability in the Netherlands specifically. In addition note that, as seen in Section 3.1.4, Section 3.2.4 and Section 3.3.4, S3 buckets are not the only cloud storage instances that can be misconfigured.

3.4.1 Warnings found in S3 buckets

One of the first things that was noticed after executing

```
python3 bucket-stream.py
```

was that an S3 bucket containing backups was exposed. In the file names of the S3 bucket a file named "poc.txt" was discovered (the bucket had only been discovered a few seconds ago). This is usually a signal that a security researcher has discovered the exposed bucket because a PoC or proof of concept is often used by security researchers to check for the existence of a vulnerability. Indeed, checking the contents of poc.txt provided us with the following message:

```
Hello from https://www.twitter.com/random_robbie - this is a proof of concept
to check if your S3 bucket has incorrect permissions.
Please secure your s3 bucket before a bad guy finds it!!
DMs are open if you wish to chat.
https://www.openbugbounty.org/researchers/Random_Robbie/
(little overview of me)
```

3.4.2 Potentially missed buckets

Because of a mistake I made when using a tab opener extension [82] (I closed the extension while it was still in the middle of opening tabs for each of the domains that was specified), it is possible that I might have missed sensitive data in the first 50 buckets from the log file that was generated by bucket-stream, but I decided to focus on the third experiment with the top 25,000 domains in Section 3.3 instead of having another look at the first 50 buckets.

3.4.3 Partially downloading ZIP files

Instead of downloading the 100 GB databases, which would have been unethical, we examined the possibility of partially downloading the ZIP files containing the databases. A quick Google search seems to suggest that although it is possible to list the file names of a partially downloaded ZIP file, it is not possible to extract specific files from it because the central directory of a ZIP, which is needed to extract a ZIP file, is located at the end of the file [83], so this option was not explored further.

3.4.4 Access denied messages

While manually reviewing the discovered buckets, we encountered a large number of "Access Denied" error messages. Although we did not keep track of the number of times this message was encountered, it does indicate that the buckets that were discovered by bucket-stream were not supposed to have been public because the settings were changed between the first time the bucket was discovered and the time that our review of the buckets started.

3.4.5 Findings live monitoring of buckets

After having executed the tool for 50 days, we found 3,241 open buckets. 47 of those buckets were still open and contained sensitive information when we started reviewing them a couple of days after bucket-stream had finished and when we were manually going through each of the log entries.

Two of the buckets contained database backups that were over 100 GB in size and according to Matthijs Koot from the Dutch Institute for Vulnerability Disclosure, one of these buckets had already been online for multiple years (this was discovered by using the buckets.grayhatwarfare.com website mentioned in Section 2.4.3).

The results from this experiment are comparable to the results from Section 6.3 of [14] which found 40 Azure Blobs (not S3 buckets) containing personally identifiable information, but that paper used a set of 13,705 readable buckets as opposed to only 3,241 buckets and a brute force approach combining MassDNS [19] and wfuzz [84]. This seems to suggest that S3 buckets contain sensitive information more often than Azure Blobs because despite the fact that we looked at 10,464 fewer readable buckets than [14] did, we were still able to find 7 more storage instances that contained sensitive information than [14].

It is also possible that our approach was more successful at identifying sensitive instances because monitoring for misconfigured cloud instances through certificate transparency logs could be a more effective approach than using brute force.

This seems plausible because while monitoring for these instances, we use keywords such as backup and develop, which are often associated with the storage of sensitive information, and we would identify storage instances fairly quickly after they are exposed publicly. [14] might have used similar keywords, however, as Section 6.4 of that paper seems to imply.

3.5 Evaluation of findings

In this section, we compare the results from the experiments in sections 3.1, 3.2, 3.3 and 3.4 to the results from prior work.

3.5.1 .git directories (I)

The research on the exposure of .git directories by Internetwache [2] found 9,700 publicly accessible Git repositories by scanning the Alexa Top 1M for this issue. In sections 3.1.2, 3.2.2 and 3.3.2, we found 789 public accessible Git repositories by scanning 5,998+44,553+308,141 = 358,692 subdomains for this issue (done in three phases by first targeting subdomains of the top 25, then the top 1,000 and finally the top 25,000 domain names). In terms of root domain names, we found that 349 out of the 25,000

(1.4%) were affected. These findings seem to correspond to the ones from Internetwache in 2015 and according to the findings from Internetwache [2], there was a correlation between the probability of finding a website that was affected by exposed .git directories when looking at less popular domains in the Alexa Top 1M. Combined with the fact that less than 1% of the Alexa Top 1M was identified to be vulnerable to this issue, it seems like including subdomains when looking for exposed .git directories might increase the probability of finding them, but this does assume that the research by Internetwache and our own research was done correctly.

3.5.2 Misconfigured cloud storage (III)

As noted at the end of 3.4, [14] found 40 Azure Blobs containing personally identifiable information (PII) and we found 47 S3 buckets containing PII. Note that we did use a different research methodology and of course these findings concern different cloud providers.

3.5.3 Comparison findings in prior work and new findings

Table 1 provides a comparison between our findings and those of prior work.

No prior research seems to exist about dependency files, so the findings from Section 3.3.3.4 and Section 3.3.3.6 are hard to evaluate, but we did show that the leaking of dependency files is an issue that occurs more frequently than we thought at first based on the results from Section 3.2.3 and Section 3.3.3.

Furthermore, we did not find research (in the academic literature) which used the cloud_enum tool to check a number of different cloud providers for information leaks. Table 2 summarizes our new findings.

3.5.4 Exposed databases (IV)

Finally, in Section 3.1.5 we found more than 10,000 Redis servers, 18,225 Elasticsearch instances, and many compromised MongoDB instances that were exposed to the Internet (we did not look at the exact number of compromised instances for the last point).

My thesis	**Automating Bug Bounty** [14]
47 sensitive S3 buckets exposed	40 sensitive Azure Blobs exposed
My thesis	**Internetwache** [2]
1.4% of top 25,000 Dutch domains leak .git directories	<1% of Alexa Top 1M leak .git directories
My thesis	**Open .git global scan** [85]
1.4% of top 25,000 Dutch domains leak .git directories	<1% of 230,000,000 websites leak .git directories

Table 1: Comparison of quantitative findings

Dependency files
At least 5.96% of the top 13,567 domains use vulnerable dependencies (3.3.3.6)
Cloud storage instances
60 DigitalOcean spaces provided LIST permissions to anonymous users (no sensitive information). (3.1.4.2)
8 Firebase databases leaked sensitive information (3.2.4.2)
One Google Cloud storage instance leaked sensitive information (3.2.4.2)

Table 2: Additional findings from this thesis

4 Future work

The future work section is split into two subsections. Section 4.1 describes the improvements that could be made to our research and our methodology. Section 4.2 describes related research areas and ideas that we think could be interesting to future researchers.

4.1 Research improvements

There are a number of limitations in our research. Below are some suggestions that could help when improving this research.

One of the reoccurring themes in my experiments was the trade-off between the thoroughness of my tests and the speed of the tests. Being more thorough usually means sending more network traffic, which can slow down the gathering of results. Being less thorough means that the gathering of results can go faster, but it can also increase the number of false negatives. The tension between these two aspects was one of the main problems that I came across while conducting the experiments in sections 3.1 to 3.4.

4.1.1 Subdomain enumeration

An alternative approach to subdomain enumeration could use permutations such as the ones generated by [86] and brute forcing. The reason we did not use brute-forcing was that the amount of time it would take to do so made it infeasible. Another contribution that could be made is increasing the space efficiency of subdomain enumeration tools. Finally, more research could be done about the quality of passive data sources (i.e, how many of the subdomains that they return are valid) and scripts could be written to use multiple data sources with a single tool [87].

4.1.2 .git directories (I)

Research into subversion (.svn) directories could also prove interesting, although our guess is that these types of issues are likely to become less common as vulnerability scanners are already checking for them [88] and even the original research into .git directories by Internetwache.org did not find a large number of affected domains. On the other hand, our research results show that this is still an interesting area. Other types of directories that could be checked for (as pointed out by a Reddit comment [89]) are .bzr directories, .hg directories, and other directories used internally by version control systems. There is nothing secret about the existence of these directories, but as long as web developers accidentally expose them, security researchers should keep reporting this issue to help prevent malicious attackers from extracting the source code of websites.

4.1.3 Leakage of vulnerable third-party dependencies (II)

Another interesting approach would be to research how prevalent these problems are in specific business sectors such as the banking sector, government sector, or in academic institutions. Based on the results of such research, a vulnerability scanner could be implemented that uses fuzzing techniques that are targeted at specific dependencies and their associated issues.

Additionally, the effectiveness of multiple dependency checker tools could be compared as was done in [90]. The MergeBase article only covers .NET and Nuget projects and independent research that replicates the results listed in that article as well as research

that compares the effectiveness of tools that are written for other languages such as Rust or Go could be helpful for developers that are trying to decide what tool(s) to use for their own project.

As part of this research, applications inspired by DVWA (Damn Vulnerable Web Application) [91] were used to try out different dependency checkers. These applications are often used as educational projects to teach developers and security enthusiasts about common vulnerability types in applications written for different frameworks. Unfortunately, some of these educational projects do not include vulnerable dependencies in their materials. Researchers could help expand these projects to make them more comprehensive and to make it easier for other researchers to use them as a part of their toy examples.

No example vulnerable web apps seem to exist for Elixir, Rust, and Go yet. It was also noticed that some tools such as [50] depend on a relatively small vulnerability database compared to e.g [92]. Work is needed to extend the number of signatures that are used by dependency check tools.

Furthermore, while conducting this research, a new article was released which discusses dependency confusion vulnerabilities [93]. The discovery of these vulnerabilities could be integrated into the existing workflow that was described in this thesis to look for dependency confusion in the files that are discovered. This is an active research area and a project has recently been released which aims to mitigate dependency confusion vulnerabilities [94].

4.1.4 Discovering misconfigured cloud storage (III)

No public tools seem to exist, to the best of our knowledge, to check for misconfigured Alibaba Object Storage Service or Oracle Cloud Object Storage services despite the fact that a workflow to do so has been published for both of these by [14]. A new open-source tool could be developed for this in a paper.

Additionally, a tool that combines the checking for every common cloud storage service would be ideal because it would allow researchers to simply their workflows. cloud_enum already tries to accomplish this and it could be expanded to support DigitalOcean, Alibaba, and Oracle Cloud storage services.

Because Section 3.2.4.2 only resulted in finding misconfigured Google Cloud platform instances, it could be interesting to look further into Google Cloud security specifically. A number of ideas about this area can be found in a talk at the BugCrowd LevelUp security conference [95].

4.1.5 Discovering exposed databases (IV)

There are a wide variety of different database management systems (DBMS). A paper discussing what databases allow for a configuration where no authentication is required, what ports they listen on, and how the absence of authentication can be detected could be useful to get more accurate estimations of the number of publicly exposed databases as it allows researchers to know exactly how to look for these systems.

4.1.6 Web Application Firewall notifications

In the initial draft of this research, we wanted to incorporate Slack notifications to warn us when a domain would block our requests. This, together with IP rotation, could help to increase the number of vulnerabilities that would be discovered. Because we only had access to a single virtual private server, however, this option was not explored.

4.1.7 Logging of all network timeouts

The research in this thesis involved sending lots of network packets. Inevitably some of those packets got dropped and connections to websites sometimes timed out when brute-forcing for container names on Azure Blobs, for example. Hence, a comprehensive logging system could be implemented when performing this type of research to increase the reliability of research results and decrease the number of false negatives when searching for vulnerabilities.

4.1.8 Writing to cloud storage

In our research, we did not look at the possibility of writing to discovered misconfigured cloud storage instances because it is not directly related to information disclosure vulnerabilities, but publicly writable cloud instances can have many legal implications as well as security implications. If a cybercriminal uses a cloud storage instance to store illegal content such as pirated movies or overwrites JavaScript files as can be done in Magecart attacks [15], the consequences could be dire. As the `poc.txt` files discovered in Section 3.4.1 seem to demonstrate, this issue often exists in publicly readable S3 buckets, so it could be interesting to research publicly writable S3 buckets using similar approaches to the ones used in this thesis.

4.2 Related research areas

There are many research areas that went unexplored in this thesis. Below are a few ideas that future research could explore.

4.2.1 API keys in continuous deployment build logs

Similar to how API keys get leaked in source code repositories, they also can end up in the build logs of a continuous deployment system. An example of this can be found in [96] and, as discussed in the blog post, a tool that allows you to search through Travis-CI build logs can be found at [97].

4.2.2 Hardcoded API keys in mobile apps

A third location where API keys could get leaked is in mobile apps. Media coverage about this issue includes [98].

4.2.3 Spring Boot Actuators

Spring Boot Actuators are used to configure applications that are built on the Spring Boot Framework. The first time I was exposed to Spring Boot Actuators was through an exercise on PentesterLab [99] and it was based, in turn, on an article from Veracode [100]. A future paper could check for the presence of "/env" on multiple domains to determine how many Spring Boot applications do not properly limit access to their actuators. Caution is advised when looking at this issue because you could gain access to confidential information this way.

4.2.4 Misconfigured content management systems

As part of his bug bounty journey, the author of this thesis once found a Magento webshop that leaked customer information because of a misconfiguration discovered through magescan [101] (of course, the issue has now been fixed).

Though using vulnerability scanners could be a bit too aggressive for security research if permission has not been granted to use them, misconfigured content management systems offer another attack vector for information leakage.

The Dutch Institute for Vulnerability Disclosure explored this area in the Netherlands and found multiple vulnerabilities in 313,000 WordPress sites [102]. Although it is not explicitly mentioned how many vulnerabilities were discovered, around 100 notifications were sent to website owners for two major vulnerability types.

4.2.5 Default credentials

One area that we have not explored is default credentials. The reason for this is because the law on Computervredebreuk (which details some types of computer crime) in the Netherlands makes it illegal to gain unauthorized access to a system when you have not been provided explicit permission to do so. The author of this thesis is not a lawyer, but based on this observation it was deemed too risky to check for the presence of default credentials. In consultation with a lawyer, the reader might discuss whether research into this area is feasible in their own country.

4.2.6 Continuous monitoring

Projects such as Assetnote [103] have started to provide commercial offerings to help organizations monitor their attack surface. It could be interesting to investigate Assetnote as well as other products and to check what areas have not been included in continuous monitoring yet because the reason they have not been included yet could be because more research is needed in those areas.

4.2.7 Scalable approaches to automation using distributed computing

In the presentation Bug Bounty Hunting on Steroids [104], a group of bug bounty hunters demonstrated a tech stack based on Golang, Docker, Kubernetes, and Argo that could be used for a more efficient implementation of the workflows discussed in this thesis. Alternative tech stacks could be investigated for the construction of the most optimal workflow.

One open source tool that seems promising in this area, but does not use many of the aforementioned technological solutions is Axiom. A demo of it can be found at [105] and it could have potentially allowed us to execute some of our research steps in minutes instead of hours. It can do this because of the automatic scaling of processes across multiple virtual private servers. At the time of this writing, DigitalOcean, IBM Cloud, Linode, and Azure are officially supported providers [106].

4.2.8 Universally accepted sampling methods

For practical reasons, we used a method based on stratified sampling to limit the number of subdomains that needed to be examined. Universally accepted sampling methods should be established to determine whether this is a correct approach.

4.2.9 Formal methods to determine the quality of information disclosure research

Security research is not yet a fully formalized discipline. A list of metrics that the quality of information disclosure research can be measured by could be useful to determine how much a paper contributes to the security research space.

4.2.10 API keys in GitHub repositories (V)

The most creative aspect of looking for API keys in GitHub repositories is probably the method that is used to detect them because there are a wide variety of services for which an API key can be generated and not every detected API key is sensitive. An automated system based on keyhacks (listed in the references) could be developed to check for the exploitability of the discovered API keys, but the ethics of such research are questionable (because it requires using API resources that you do not have permission for). Hence, this option should only be explored in a theoretical setting and after discussions with an ethics committee. One legitimate use case of such a system would be that it could be used to reduce the number of false positives in static analysis systems that warn for hardcoded API keys (although this might incur too much of a performance cost).

4.2.11 Vulnerable Virtual Private Network solutions

In the past there have been a number of incidents where the passwords of Virtual Private Networks got exposed, potentially allowing an attacker to access the internal networks of an organization [107]. It could be interesting to scan the Internet and check how many VPNs are still vulnerable to these types of issues. The owners of the VPNs should then be notified of the issue.

5 Defenses against information disclosure vulnerabilities

In this section, we describe both technical and organizational measures that could be taken to help improve the security against information disclosure vulnerabilities.

This section is not intended to provide an exhaustive list of protections against these types of vulnerabilities and implementing the suggestions listed here will not protect you against every type of information disclosure vulnerability. We merely intend to describe how to protect against the five types of information disclosure vulnerabilities that we used as a baseline in this thesis, but as is demonstrated in Section 4 there are many more types of information disclosure vulnerabilities that exist, and we have undoubtedly not described every type of information disclosure vulnerability in that section either.

We strongly recommend that you think about how the advice given in this section fits into your own security policies and that you do not only copy the suggestions provided below.

5.1 Defenses against exposing subdomains

As far as we are aware, it is not possible to prevent someone from enumerating the subdomains of a domain, similar to how it is not possible to prevent someone from using port scanning to find what services you expose on a server. Hence, the best way to protect against the risks posed by subdomain enumeration is to protect against associated attacks such as subdomain takeovers and to not expose your internal systems to the Internet.

The use of DNS wildcards can make subdomain enumeration more annoying because all subdomains will resolve including subdomains not hosting real applications, but this method should not be depended upon because the IP addresses that wildcard subdomains resolve to can be filtered out (for an example of this approach, see [108]).

5.1.1 Defenses against subdomain takeovers

To protect against subdomain takeover attacks there are a number of monitoring tools that could be used.

Subdomain takeovers are caused by human error. For example, a developer might setup a subdomain pointing to a third party service, and when the subscription to that service expires and the DNS record pointing to it is no longer valid, an attacker can create their own account for the third party service and take over the contents of the subdomain.

To prevent this from happening, it is critical that such subdomains are discovered as quickly as possible. There are a number of commercial asset monitoring products that can help with this ([103] and [109]) and there are also custom setups that could be used [110]. Many of these setups depend on fingerprints matching web pages of third party services containing messages such as "this subdomain is not registered", but subdomain takeovers can also be caused by DNS misconfigurations not involving third party services [111].

Once a subdomain takeover is discovered, the incorrect DNS record(s) can be removed. Next to monitoring, security awareness training for developers could help when protecting against this issue as well.

5.1.2 Defenses against the exposure of internal systems to the Internet

The author of this thesis has discovered multiple instances of internal systems being exposed to the Internet while participating in bug bounty programs. This type of issue can occur when a system is setup without requiring any type of authentication and no firewall rules are in place to prevent the system from being reachable externally.

Performing port scanning or using vulnerability scanners might allow employees on a security team to detect these types of issues before an attacker discovers them, but security awareness is probably the most important measure to protect against this type of situation because developers might expose a system on the Internet without knowing about the risks that are involved in doing so.

Ideally, there should be a corporate policy that explains in what situations it is acceptable to expose an internal system to the Internet and how this can be done "securely" (for example, by only making the system accessible through a VPN and by requiring two factor authentication to access the system). The pros and cons of multiple policies should be considered.

5.2 Defenses against exposure of .git directories (I)

The reason that .git directories get exposed is because web servers are not configured correctly to prevent access to .git directories. One way that such issues could be protected against is by creating hardened configuration templates that can be used by people working on an operations team so that .git directories are not exposed by default.

5.3 Defenses against leakage of vulnerable third-party dependencies (II)

Using tools such as Dependabot [9] to help keep the dependencies of each of your software development projects up to date is a good step to take. This way, even if the list of dependencies that you use is exposed, at least an attacker will not be able to use any publicly known vulnerabilities to target your application. There are also organizations such as Snyk [112] which provide services to help with this. To prevent the leakage of dependency manager files, it is possible to use a blocklist in your web server configuration that prevents people from requesting them similar to how you could block .git directories as described in Section 5.2.

5.4 Defenses against misconfigured cloud storage (III)

This thesis only looked at the public disclosure of files in cloud storage instances. An excellent blog post that describes a number of ways that you could prevent this on Amazon Web Services is [113]. On Microsoft Azure, there is a setting that allows you to completely disable the possibility of public blobs [114]. Google Cloud Platform also provides options to prevent the leakage of data in storage buckets [115].

5.5 Defenses against exposed databases (IV)

Similar to Section 5.1.2 a combination of monitoring and having clear guidance on the secure configuration of databases that should be followed by every employee could help to prevent the exposure of databases. Additional advice can be found at [116].

5.6 Defenses against API keys in GitHub repositories (V)

A combination of security awareness training and using tools such as detect-secrets [117] or GitHub secret scanning [118] to monitor for the leakage of secrets is one way that an organization could protect against this vulnerability.

5.7 Steps forward

Although the vulnerabilities that have been researched in this thesis are well-known, it quickly became clear to me in the responsible disclosure process that there are many security professionals who do not understand the impact that some of the vulnerabilities described above can have on an organization.

For example, the exposure of .git directories can potentially allow an attacker to find vulnerabilities using the source code of an application more easily than they would be able to without access to source code (for the differences between whitebox and blackbox testing, see [119]), and yet some organizations thought that because they did not have sensitive API keys that were stored in their source code, this issue was not that serious.

Furthermore, both the research by Internetwache [2] and the research by Vladimir Smitk [85] found that less than 1% of domains that were scanned were affected by this issue whereas my research found that 1.4% of 25,000 Dutch domains were affected by this issue. This seems to indicate that domains from the Netherlands are more commonly affected than other domains, but other research also looked at many more domains (around 1,000,000 in the research by Internetwache and 230,000,000 in the research by Vladimir Smitk).

It was also surprising how many websites were using outdated dependencies and were leaking the fact that they did so (see Section 3.3.3.4).

I think one big step forward would be the publication of documents that are targeted towards employees working in IT and that discuss the many information disclosure risks that exist and how to protect against them (e.g, the fact sheets of the National Cyber Security Centre could be updated to include information about this attack vector).

Furthermore, the research done by groups such as the Dutch Institute for Vulnerability Disclosure could help with improving our national digital security posture and, given the right funding, it might be possible to set up monitoring systems that can help to discover these types of issues in real time, so the affected organizations are no longer dependent on a group of independent security researchers or bug bounty hunters to find low hanging fruit vulnerabilities and instead the DIVD could report these issues.

Of course, the DIVD is just one institute that performs this type of research, but if automation could be used to protect countries against a set of basic vulnerabilities, I think this could raise the bar for malicious attackers. A proactive approach to cyber security seems to be lacking at the moment.

6 Conclusions

As can be seen from this thesis, there are many ways in which information disclosure vulnerabilities can present themselves in the software development process of closed source web applications and as can be seen in Section 4, we only scratched the surface in our research.

The most interesting findings from the three sets of experiments that I did, were that 1) around 5.96% of the top 13,567 Dutch domain names used dependencies with known vulnerabilities in them, 2) 1.4% of the top 25,000 Dutch domain names leaked the contents of .git directories and 3) using the bucket-stream [54] tool, I was able to find at least 47 S3 buckets that contained sensitive information (for more findings, see Section 3.5.3).

I find these findings interesting not because they detail new types of security vulnerabilities, but because despite the fact that most of these issues have been discussed multiple times and shown up in both academic and industry publications, they continue to be a problem that affects a substantial number of organizations. As long as these types of vulnerabilities continue to exist, the bar for opportunistic cyber criminals is set very low.

The bug bounty hunters on the top of the leaderboard of bug bounty platforms like BugCrowd and HackerOne have become skillful at automatically finding vulnerabilities at a large scale (as demonstrated by the hundreds of submissions that they do every year), and while their intentions are usually good, cyber criminals can apply the same business model to a much larger attack surface.

Although many databases were found to be exposed without authentication, there were so many of these databases that were already tagged as compromised in Shodan [62], that we decided not to explore this area because we thought that reporting these compromises (most were based on ransomware) would not do much to help the affected organizations (see Section 3.1.5). We did not explore exposed API keys on GitHub either because of the reasons discussed in Section 2.6.6.

As discussed in Section 3, the experiments were performed in three steps: First the top 25 domains of the Netherlands were analyzed, then the top 1,000 domains were analyzed, and finally the top 25,000 domains were analyzed. The rankings of domains were determined based on a custom generated Tranco list [73] [75]. Each of the sections 3.1, 3.2 and 3.3 already have a subsection containing conclusions, but for the convenience of the reader we have combined all our final conclusions in Section 6.

6.1 Conclusions about first experiment

The first experiment on the top 25 domains of the Netherlands showed that the method of using passive subdomain enumeration is quite effective, but also that the domains that include dynamic IP addresses can cause a bad distribution (i.e, if we were to randomly pick a number of subdomains to include in our experiments from the results of passive subdomain enumeration tools, we would often end up testing the subdomains that include dynamic IP addresses).

It also showed that proper sampling methods were needed to ensure we could look at the same number of subdomains for each root domain. The need for proper sampling methods was further underlined by the fact that some root domains had hundreds of

thousands of subdomains.

When looking for .git directories, we found that two out of the 25 organizations that were examined leaked the source code of their applications. We did not yet warn the affected organizations because we were still trying to determine what would be the best way to responsibly disclose our findings. We are currently in the process of disclosing all our findings.

No vulnerable third-party dependencies were identified and we also did not find any dependency manager files that were leaked. No misconfigured cloud instances were found (60 DigitalOcean spaces had LIST permissions, but they did not contain important information) and we discovered that we could only use about 14 keywords to efficiently look for them on a scale of 25,000 domains using a single virtual private server.

In the first experiment, we also gave up on looking for exposed databases because almost all databases allowing unauthenticated public access, as discovered through Shodan, were already compromised.

Finally, we did not look at the leakage of API keys in GitHub repositories because of the prior research that already exists surrounding this topic and more importantly because of the financial and computational costs associated with using many keywords.

6.2 Conclusions about the second experiment

In the second experiment on 1,000 target domains, we collected 32,735 resolvable subdomains to examine. We found 25 more organizations that leaked source code in .git directories (meaning 27 out of 1,000 organizations were affected so far) and, like in the previous experiment, no vulnerable third-party dependencies were identified.

The identified misconfigured cloud instances were often supported by Google Cloud platform, but a surprisingly low number of vulnerable storage instances were discovered, only nine. With such a small number of findings, I am not ready to make the claim that Google Cloud instances are misconfigured more often than Azure instances or Amazon Web Services instances.

Based on these results, we decided to change the strategy that was used to look for vulnerable third-party dependencies in the third experiment, and we also expanded the cloud services to look for in the third experiment by including Alibaba Object Storage services and Oracle Cloud storage services.

6.3 Conclusions about the third experiment

Neither the research on Alibaba Object Storage nor the research on Oracle Cloud storage services led to any vulnerable instances being identified, and because we discovered 349 domains that were leaking their source code in .git directories, we decided to prioritize the disclosure of these findings instead of manually investigating more cloud instances (the email templates that we used for the disclosure of our findings can be found in Section A.2).

In Section 3.3.3.6 we also found 809 domains that used dependency files with known vulnerabilities and this had to be disclosed to the affected organizations as well.

It was shocking to me that despite the fact that the exposure of .git directories is an issue that has been known about for quite some time (going back to at least 2015), there were still numerous government websites as well as websites belonging to well-known organizations in the Netherlands, that were leaking their source code in this way. Sometimes, even the source code of the main `www` domain belonging to an organization was leaked and there were also a number of websites that leaked sensitive tokens in `/.git/config` files.

In Section 3.4 we found 47 S3 buckets that had sensitive information in it, even though it took a number of days before we started reviewing the log file that was generated by the bucket-stream tool (also see Section 2.4.3). This is comparable to the results from [14] (for more comparisons between the results of this thesis and other research as well as a summary of our findings, see Section 3.5) and seems to indicate that although the monitoring of S3 buckets is still a viable business model for cybercriminals, this attack vector might become less interesting in the future (especially given the number of hours that it takes to review the discovered S3 buckets).

6.4 Responsible disclosure

At the end of our research, we disclosed our government-related findings to the National Cyber Security Center (NCSC) and we are in the process of disclosing the specific vulnerabilities that we found to each affected organization. We expect to finish reaching out to all organizations in the upcoming weeks. Unfortunately, due to time constraints, it was not possible to fully finish the responsible disclosure process before submitting this thesis. We advice the readers of this thesis to check whether their own organization is protected against these vulnerabilities.

So far, I have done at least 21 submissions (some of which were about multiple domain names such as my reports to the NCSC and the informatiebeveiligingsdienst [120]), which were mostly to organizations with a brand name that I recognized or to organizations that I considered of vital importance (government institutions and hospitals). I am not sure about the exact number of submissions I have done because some companies only allow you to contact them via their website, but I do keep track of a list of companies I still need to report vulnerabilities to.

All organizations that responded did respond positively, but there are still eight organizations that never got back to me. Furthermore, one of the reasons that the responsible disclosure process takes a while is that most organizations do not yet have a responsible disclosure policy or a security email address, which means most of my reports have to go through a customer support channel first (and this process can take multiple weeks).

At first, I wanted to cooperate with Open Bug Bounty [121] to speed up the disclosure process (Open Bug Bounty is a non-profit created by independent security researchers that has responsibly disclosed over 900,000 vulnerabilities at the time of this writing), but they no longer accept the types of vulnerabilities that were discovered in this thesis (possibly because they got too many vulnerability reports related to them). In general, it has been difficult to get organizations to help me in the submission process, and I have had to report most of my findings on my own for now.

The misconfigured cloud findings have been reported to the cloud security teams working for cloud providers so they can inform their customers about the issues.

6.5 Defenses against information disclosure vulnerabilities

In Section 5 we discuss a number of defenses against information disclosure vulnerabilities and the role that organizations such as the National Cyber Security Center can play to improve the security posture of the Netherlands against these basic types of attacks.

6.6 Reflection on the research process

- Finding the right balance between the time spent on hands-on research and academic writing can be tricky and seems best learned from experience.
- Security research still has much room to improve, especially when it comes to the area of formalization (as mentioned in Section 4.2.8 and Section 4.2.9).
- Good communication between a student and their supervisors is vital for a successful research project.
- Writing down how much time a certain step took immediately after you have performed it is helpful.
- Having multiple research areas can help prevent situations where a research project fails because of one idea that ends up not being feasible.
- Your research topic can change a few times during your research.

6.7 Reflection on findings

- Old mistakes tend to stay around in security because humans will continue to make the same mistakes (see conclusions in Section 6.3. It was shocking to me how many popular domains still leak their source code).
- Looking for API keys on GitHub is expensive on a large scale, which makes it infeasible for our research (see Section 2.6.6).
- InfoSec people like to advertise themselves by writing text files to insecure S3 buckets (see Section 3.4.1).

6.8 GitHub repository

A GitHub repository containing the custom scripts used in this research can be found at [1]. It also contains a word list that is based on the active subdomains of the top 25,000 most popular domains in the Netherlands.

References

[1] Maurice Dibbets. *Scripts used for thesis*. May 17, 2021. URL: https://github.com/oxygen005/thesis.

[2] Internetwache.org. *Don't publicly expose .git or how we downloaded your website's sourcecode - An analysis of Alexa's 1M*. July 28, 2015. URL: https://en.internetwache.org/dont-publicly-expose-git-or-how-we-downloaded-your-websites-sourcecode-an-analysis-of-alexas-1m-28-07-2015/.

[3] sahiltikoo. *Information disclosure on https://paycard.rapida.ru*. Jan. 20, 2018. URL: https://hackerone.com/reports/299552.

[4] Help Net Security. *Misconfigured cloud storage services are commonplace in 93% of deployments*. Aug. 6, 2020. URL: https://www.helpnetsecurity.com/2020/08/06/misconfigured-cloud-storage-services/.

[5] Catalin Cimpanu. *Hacker ransoms 23k MongoDB databases and threatens to contact GDPR authorities*. July 1, 2020. URL: https://www.zdnet.com/article/hacker-ransoms-23k-mongodb-databases-and-threatens-to-contact-gdpr-authorities/.

[6] Catalin Cimpanu. *Over 100,000 GitHub repos have leaked API or cryptographic keys*. Mar. 21, 2019. URL: https://www.zdnet.com/article/over-100000-github-repos-have-leaked-api-or-cryptographic-keys/.

[7] OWASP. *OWASP Top 10 2017*. Nov. 2017. URL: https://owasp.org/www-project-top-ten/2017/Top_10.html.

[8] M. Campbell. *Keep your dependencies secure and up-to-date with GitHub and Dependabot*. Jan. 31, 2019. URL: https://github.blog/2019-01-31-keep-your-dependencies-secure-and-up-to-date-with-github-and-dependabot/.

[9] Dependabot. *Dependabot*. URL: https://github.com/dependabot/dependabot-core (visited on 01/28/2021).

[10] Michael Meli, Matthew R. McNiece and Bradley Reaves. "*How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories*". In: Proceedings of the 26th Annual Network and Distributed System Security Symposium. NDSS 2019. Feb. 2019. DOI: 10.14722/ndss.2019.23418.

[11] Dutch Institute for Vulnerability Disclosure. *DIVD contact form*. URL: https://www.divd.nl/divd-nl/contact/ (visited on 02/26/2021).

[12] National Cyber Security Centre. *Coordinated Vulnerability Disclosure: the Guideline*. June 1, 2019. URL: https://english.ncsc.nl/binaries/ncsc-en/documents/publications/2019/juni/01/coordinated-vulnerability-disclosure-the-guideline/WEB_Brochure-NCSC_EN.pdf.

[13] Openbaar Ministerie. *Coordinated Vulnerability Disclosure / ethisch hacken*. Jan. 29, 2021. URL: https://www.om.nl/onderwerpen/cybercrime/coordinated-vulnerability-disclosure---ethisch-hacken.

[14] Ján Masarik. *Automating Bug Bounty*. Master thesis, Masaryk University. 2019. URL: https://is.muni.cz/th/de05t/master_thesis_final.pdf.

[15] Yonathan Klijnsma. *Spray and Pray: Magecart Campaign Breaches Websites En Masse Via Misconfigured Amazon S3 Buckets*. July 11, 2019. URL: https://www.riskiq.com/blog/labs/magecart-amazon-s3-buckets/.

[16] Appsecco. *The Art of Subdomain Enumeration*. Apr. 1, 2019. URL: https://appsecco.com/books/subdomain-enumeration/.

[17] Jason Haddix. *The Bug Hunter's Methodology v4.0 - Recon Edition by @jhaddix #NahamCon2020!* June 19, 2020. URL: https://www.youtube.com/watch?v=p4JgIu1mceI.

[18] OWASP. *Amass*. URL: https://github.com/OWASP/Amass (visited on 02/26/2021).

[19] B. Blechschmidt. *MassDNS*. URL: https://github.com/blechschmidt/massdns (visited on 02/26/2021).

[20] Project Discovery. *Fast passive subdomain enumeration tool*. URL: https://github.com/projectdiscovery/subfinder (visited on 03/21/2021).

[21] Project Discovery. *httpx*. URL: https://github.com/projectdiscovery/httpx (visited on 04/02/2021).

[22] BinaryEdge. *Pricing - BinaryEdge*. URL: https://www.binaryedge.io/pricing.html (visited on 04/02/2021).

[23] Censys. *Censys*. URL: https://censys.io/account/billing (visited on 04/03/2021).

[24] Opsmate. *Cert Spotter*. URL: https://sslmate.com/certspotter/api/pricing (visited on 04/03/2021).

[25] Farsight Security. *DNSDB Community Edition*. URL: https://www.farsightsecurity.com/dnsdb-community-edition/ (visited on 04/03/2021).

[26] GitHub. *Resources in the REST API*. URL: https://docs.github.com/en/rest/overview/resources-in-the-rest-api (visited on 04/03/2021).

[27] Intelligence X. *Intelligence X*. URL: https://intelx.io/ (visited on 04/03/2021).

[28] RiskIQ. *RiskIQ Community Edition*. URL: https://community.riskiq.com/settings (visited on 04/03/2021).

[29] Recon.Dev. *Recon.Dev - Recon Simplified*. Aug. 13, 2020. URL: https://recon.dev/pricing.html.

[30] Robtex. *Robtex API*. July 22, 2017. URL: https://www.robtex.com/api/.

[31] SecurityTrails. *Home — SecurityTrails Account Dashboard*. URL: https://securitytrails.com/app/account?plan=api-0 (visited on 04/03/2021).

[32] Shodan. *Shodan Credits Explained*. Nov. 2, 2020. URL: https://help.shodan.io/the-basics/credit-types-explained.

[33] Johannes Gilger. *Quotas - urlscan.io*. URL: https://urlscan.io/products/api/ (visited on 04/03/2021).

[34] VirusTotal. *Quotas*. 2019. URL: https://developers.virustotal.com/v3.0/docs/quotas.

[35] Spyse. *Plan & Pricing — Spyse*. URL: https://spyse.com/pricing (visited on 04/02/2021).

[36] ZoomEye. *ZoomEye - Cyberspace Search Engine*. URL: https://www.zoomeye.org/business (visited on 04/03/2021).

[37] OWASP. *Amass configuration file*. URL: https://github.com/OWASP/Amass/blob/master/examples/config.ini (visited on 02/26/2021).

[38] GitHowTo. *Inside Git: .Git directory*. URL: https://githowto.com/git_internals_git_directory (visited on 03/21/2021).

[39] Marie M. *Source code disclosure via exposed .git folder*. Oct. 25, 2018. URL: https://pentester.land/tutorials/2018/10/25/source-code-disclosure-via-exposed-git-folder.html.

[40] N. Anantharaman and B. Wukkadada. "*Identifying The Usage Of Known Vulnerabilities Components Based On OWASP A9*". In: 2020 International Conference on Emerging Smart Computing and Informatics (ESCI). 2020, pp. 88–91. DOI: 10.1109/ESCI48226.2020.9167645.

[41] I. Pashchenko, D. -L. Vu, and F. Massacci. "*Preliminary Findings on FOSS Dependencies and Security : A Qualitative Study on Developers' Attitudes and Experience*". In: 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). 2020, pp. 284–285. DOI: 10.1145/3377812.3390903.

[42] R. Elizalde Zapata, R. G. Kula, B. Chinthanet, T. Ishio, K. Matsumoto and A. Ihara. "*Towards Smoother Library Migrations: A Look at Vulnerable Dependency Migrations at Function Level for npm JavaScript Packages*". In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2018, pp. 559–563. DOI: 10.1109/ICSME.2018.00067.

[43] A. Decan, T. Mens, and E. Constantinou. "*On the Impact of Security Vulnerabilities in the npm Package Dependency Network*". In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR). 2018, pp. 181–191. DOI: 10.1145/3196398.3196401.

[44] T. Hudson. *meg*. URL: https://github.com/tomnomnom/meg (visited on 01/24/2021).

[45] Sonatype Community. *auditjs*. URL: https://github.com/sonatype-nexus-community/auditjs (visited on 01/24/2021).

[46] RubySec. *bundler-audit*. URL: https://github.com/rubysec/bundler-audit (visited on 02/11/2021).

[47] OWASP. *OWASP Dependency-Check*. URL: https://github.com/jeremylong/DependencyCheck (visited on 02/02/2021).

[48] pyup.io. *Safety*. URL: https://github.com/pyupio/safety (visited on 02/11/2021).

[49] Frost Ming. *pipfile2req*. URL: https://github.com/frostming/pipfile-requirements (visited on 02/11/2021).

[50] Fabien Potencier. *Local PHP Security Checker*. URL: https://github.com/fabpot/local-php-security-checker (visited on 02/15/2021).

[51] RustSec. *cargo audit*. URL: https://github.com/RustSec/cargo-audit (visited on 02/15/2021).

[52] Sonatype Community. *Nancy*. URL: https://github.com/sonatype-nexus-community/nancy (visited on 02/15/2021).

[53] Amazon Web Services. *ListBuckets*. Sept. 30, 2020. URL: https://docs.aws.amazon.com/AmazonS3/latest/API/API_ListBuckets.html.

[54] Paul Price. *Bucket Stream*. URL: https://github.com/eth0izzle/bucket-stream (visited on 04/03/2021).

[55] UpGuard Team. *Data Warehouse: How a Vendor for Half the Fortune 100 Exposed a Terabyte of Backups*. June 27, 2019. URL: https://www.upguard.com/breaches/attunity-data-leak.

[56] Trend Micro. *Unsecured AWS S3 Bucket Found Leaking Data of Over 30K Cannabis Dispensary Customers*. Jan. 27, 2020. URL: https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/unsecured-aws-s3-bucket-found-leaking-data-of-over-30k-cannabis-dispensary-customers.

[57] initstring. *cloud_enum*. URL: https://github.com/initstring/cloud_enum (visited on 02/15/2021).

[58] Appsecco. *Spaces finder*. URL: https://github.com/appsecco/spaces-finder (visited on 02/15/2021).

[59] GrayhatWarfare. *Public Buckets by GrayhatWarfare*. URL: https://buckets.grayhatwarfare.com/ (visited on 03/21/2021).

[60] Kevin Townsend. *Misconfigured Public Cloud Databases Attacked Within Hours of Deployment*. June 10, 2020. URL: https://www.securityweek.com/misconfigured-public-cloud-databases-attacked-within-hours-deployment.

[61] Vitaly Simonovich and Sarit Yerushalmi. *Never Leave Your Cloud Database Publicly Accessible*. Oct. 14, 2020. URL: https://www.imperva.com/blog/never-leave-your-cloud-database-publicly-accessible/.

[62] John Matherly. *Shodan*. URL: https://www.shodan.io/ (visited on 02/26/2021).

[63] The MITRE Corporation. *Common Vulnerabilities and Exposures*. URL: https://cve.mitre.org/ (visited on 02/26/2021).

[64] elmahdi. *Redmin API Key Exposed In GIthub*. Nov. 25, 2020. URL: https://hackerone.com/reports/901210.

[65] moro139. *Github Token Leaked publicly for https://github.com/mopub*. Aug. 16, 2019. URL: https://hackerone.com/reports/612231.

[66] Vinoth Kumar. *JumpCloud API Key leaked via Open Github Repository*. Dec. 30, 2019. URL: https://hackerone.com/reports/716292.

[67] streaak. *KeyHacks*. URL: https://github.com/streaak/keyhacks (visited on 02/15/2021).

[68] Omar Bheda. *GitDorker*. URL: https://github.com/obheda12/GitDorker (visited on 02/15/2021).

[69] Anshuman Bhartiya. *git-all-secrets*. URL: https://github.com/anshumanbh/git-all-secrets (visited on 02/15/2021).

[70] Omar Bheda. *alldorksv3*. URL: https://github.com/obheda12/GitDorker/blob/master/Dorks/alldorksv3 (visited on 02/15/2021).

[71] GitHub. *GitHub Terms of Service*. Nov. 16, 2020. URL: https://docs.github.com/en/github/site-policy/github-terms-of-service.

[72] Google. *Google BigQuery Pricing*. URL: https://cloud.google.com/bigquery/pricing (visited on 04/02/2021).

[73] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński and Wouter Joosen. *Information on the Tranco list with ID JVWY*. Feb. 19, 2021. URL: https://tranco-list.eu/download/JVWY/1000000.

[74] mansoorr123. *Not scanning all domains*. Aug. 10, 2020. URL: https://github.com/OWASP/Amass/issues/454.

[75] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński and Wouter Joosen. "*Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation*". In: Proceedings of the 26th Annual Network and Distributed System Security Symposium. NDSS 2019. Feb. 2019. DOI: 10.14722/ndss.2019.23386.

[76] MongoDB Inc. *Default MongoDB Port*. URL: https://docs.mongodb.com/manual/reference/default-mongodb-port/ (visited on 03/21/2021).

[77] hacktivist. *Accessing Unauthenticated MongoDB Database Using Shodan*. Nov. 3, 2019. URL: https://medium.com/@greedybucks/accessing-unauthenticated-mongodb-database-using-shodan-e62acc4a2922.

[78] Nadav Avital and Ori Nakar. *New research shows 75% of 'open' Redis servers infected*. June 1, 2018. URL: https://www.imperva.com/blog/new-research-shows-75-of-open-redis-servers-infected/.

[79] Google. *Firebase*. URL: https://firebase.google.com/ (visited on 04/23/2021).

[80] Mahesh Chand. *Top 10 Cloud Service Providers In 2021*. Mar. 20, 2021. URL: https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/.

[81] eth0izzle. *Bucket Stream default permutations list*. Nov. 9, 2018. URL: https://github.com/eth0izzle/bucket-stream/blob/master/permutations/default.txt.

[82] Hagen Trinter. *Open Multiple URLs*. Feb. 24, 2021. URL: https://addons.mozilla.org/nl/firefox/addon/open-multiple-urls/.

[83] Karel Vlk and nyuszika7h. *Is there a way to download parts of the content of a zip file?* Oct. 14, 2015. URL: https://superuser.com/questions/981301/is-there-a-way-to-download-parts-of-the-content-of-a-zip-file.

[84] Xavi Mendez. *wfuzz: Web application fuzzer*. URL: https://github.com/xmendez/wfuzz (visited on 05/09/2021).

[85] Vladimir Smitk. *Open .git global scan*. 2018. URL: https://smitka.me/open-git/.

[86] S. Shah. *Altdns*. URL: https://github.com/infosec-au/altdns (visited on 02/02/2021).

[87] Patrik Fehrenbach. *Amassive Leap in Host Discovery*. Apr. 7, 2021. URL: https://www.youtube.com/watch?v=yCZqgg-GNx8.

[88] Acunetix. *SVN repository found*. URL: https://www.acunetix.com/vulnerabilities/web/svn-repository-found/ (visited on 02/26/2021).

[89] kumyco. *Source Code Of 3000+ Sites Were Leaked Because Of SVN*. 2010. URL: https://www.reddit.com/r/programming/comments/9n9mo/source_code_of_3000_sites_were_leaked_because_of/c0di49x.

[90] Julius Musseau. *Scanning .NET and Nuget projects for known vulnerabilities*. Sept. 3, 2020. URL: https://mergebase.com/blog/scanning-dotnet-net-and-nuget-projects-for-known-vulnerabilities/.

[91] Robin Wood. *Damn Vulnerable Web Application (DVWA)*. URL: https://github.com/digininja/DVWA (visited on 02/15/2021).

[92] Snyk. *Vulnerability DB*. URL: https://snyk.io/vuln (visited on 02/15/2021).

[93] Alex Birsan. *Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies*. Feb. 9, 2021. URL: https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610.

[94] John Leyden. *Linux community project aims to tackle dependency confusion attacks with easy code signing, verification*. Mar. 11, 2021. URL: https://portswigger.net/daily-swig/linux-community-project-aims-to-tackle-dependency-confusion-attacks-with-easy-code-signing-verification.

[95] Rojan Rijal. *GSuite Security: Everyone wants it but not everyone gets it*. Oct. 9, 2019. URL: https://www.youtube.com/watch?v=qdelpOxTXO4.

[96] EdOverflow. *"CI Knew There Would Be Bugs Here" — Exploring Continuous Integration Services as a Bug Bounty Hunter*. Apr. 26, 2019. URL: https://edoverflow.com/2019/ci-knew-there-would-be-bugs-here/.

[97] Corben Leo. *secretz, minimizing the large attack surface of Travis CI*. URL: https://github.com/lc/secretz (visited on 03/22/2021).

[98] Ionut Arghire. *Many Mobile Apps Unnecessarily Leak Hardcoded Keys: Analysis*. Jan. 16, 2017. URL: https://www.securityweek.com/many-mobile-apps-unnecessarily-leak-hardcoded-keys-analysis.

[99] Louis Nyffenegger. *Spring Actuators*. URL: https://pentesterlab.com/exercises/spring-actuators (visited on 03/22/2021).

[100] Michael Stepankin. *Exploiting Spring Boot Actuators*. May 1, 2019. URL: https://www.veracode.com/blog/research/exploiting-spring-boot-actuators.

[101] Steve Robbins. *Scan a Magento site for information*. URL: https://github.com/steverobbins/magescan (visited on 04/03/2021).

[102] Dutch Institute for Vulnerability Disclosure. *REPORT DIVD-2020-00008 - 313 000 WORDPRESS SITES SCANNED*. Apr. 8, 2021. URL: https://www.divd.nl/reports/2020-00008-313%20000%20Wordpress%20sites%20scanned/.

[103] Assetnote Pty.Ltd. *Assetnote - Continuous Security Across Your External Attack Surface*. URL: https://assetnote.io/ (visited on 03/22/2021).

[104] Mohammed Diaa Anshuman Bhartiya and Glenn 'devalias' Grant. *Bug Bounty Hunting on Steroids*. Aug. 11, 2018. URL: https://speakerdeck.com/bountymachine/bug-bounty-hunting-on-steroids.

[105] Ben Bidmead. *Live Recon and Distributed Recon Automation Using Axiom with @pry0cc*. Apr. 28, 2021. URL: https://www.youtube.com/watch?v=tWml8Dy5RyM.

[106] Ben Bidmead. *axiom*. URL: https://github.com/pry0cc/axiom (visited on 05/14/2021).

[107] Ax Sharma. *Passwords exposed for almost 50,000 vulnerable Fortinet VPNs*. Nov. 25, 2020. URL: https://www.bleepingcomputer.com/news/security/passwords-exposed-for-almost-50-000-vulnerable-fortinet-vpns/.

[108] Robin Wood. *DNS Reconnaissance against wildcard domains*. Oct. 24, 2012. URL: https://digi.ninja/blog/dns_wildcard_recon.php.

[109] Detectify. *Asset Monitoring tool for web app security*. URL: https://detectify.com/product/asset-monitoring (visited on 05/17/2021).

[110] Vickie Li. *Building A Subdomain Takeover Monitor*. Dec. 15, 2020. URL: https://sec.okta.com/articles/2020/12/building-subdomain-takeover-monitor.

[111] Jocelyn Chan. *DNS Hijacking – Taking Over Top-Level Domains and Subdomains*. Jan. 19, 2021. URL: https://blog.detectify.com/2021/01/19/dns-hijacking-taking-over-top-level-domains-and-subdomains/.

[112] Snyk. *Snyk Open Source Security Management*. URL: https://snyk.io/product/open-source-security-management/ (visited on 02/02/2021).

[113] Teri Radichel. *How to never have a public S3 bucket*. June 21, 2019. URL: https://medium.com/cloud-security/how-to-never-have-a-public-s3-bucket-639761508700.

[114] David Okeyode. *What?!! Public Blob Not Allowed On This Storage Account?!* June 29, 2020. URL: https://azurehangout.com/what-public-blob-not-allowed-on-this-storage-account/.

[115] Samrat Ray. *How do I prevent a data breach via personal storage buckets on GCP?* Dec. 16, 2019. URL: https://www.youtube.com/watch?v=LePbfinjgBc.

[116] Zeljka Zorz. *With database attacks on the rise, how can companies protect themselves?* Oct. 14, 2020. URL: https://www.helpnetsecurity.com/2020/10/14/securing-exposed-databases/.

[117] Yelp.com. *detect-secrets*. URL: https://github.com/Yelp/detect-secrets (visited on 05/17/2021).

[118] GitHub. *About secret scanning*. URL: https://docs.github.com/en/code-security/secret-security/about-secret-scanning (visited on 05/17/2021).

[119] Guru99. *Black Box Testing Vs. White Box Testing: Key Differences*. Mar. 26, 2021. URL: https://www.guru99.com/back-box-vs-white-box-testing.html.

[120] Informatiebeveiligingsdienst. *Informatiebeveiligingsdienst*. URL: https://www.informatiebeveiligingsdienst.nl/ (visited on 06/03/2021).

[121] Open Bug Bounty. *Open Bug Bounty*. URL: https://www.informatiebeveiligingsdienst.nl/ (visited on 06/03/2021).

A Appendices

A.1 Figures

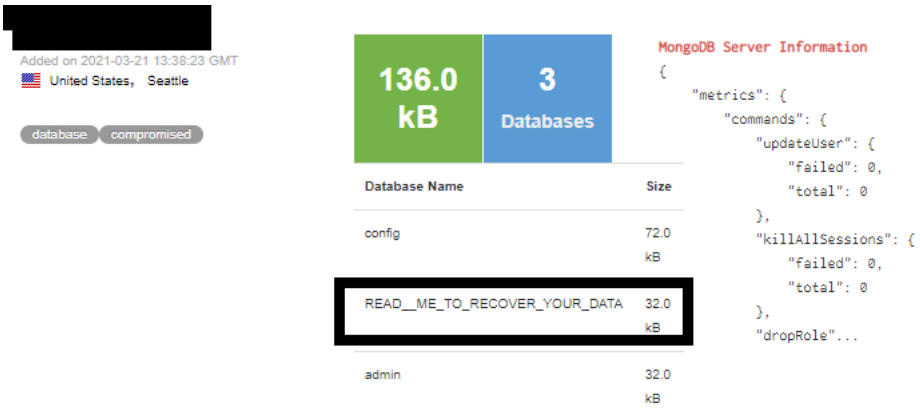

Figure 1: Compromised MongoDB instance

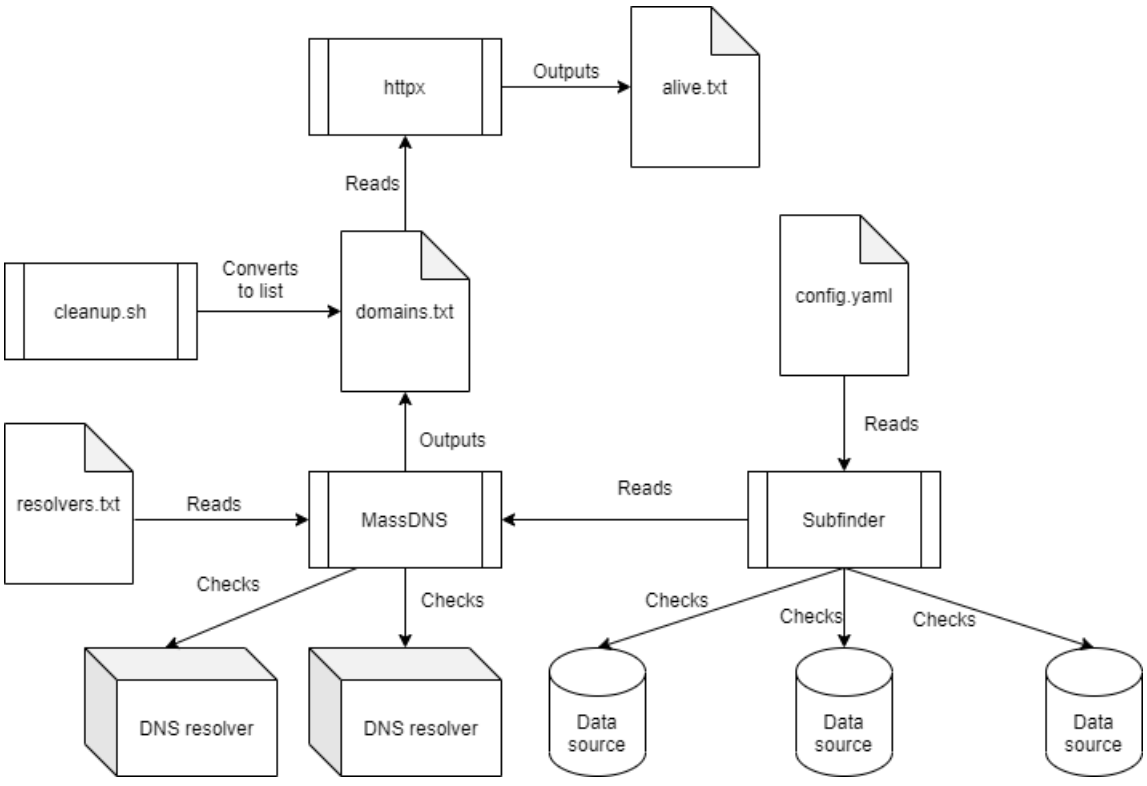

Figure 2: Workflow diagram demonstrating the subdomain enumeration process

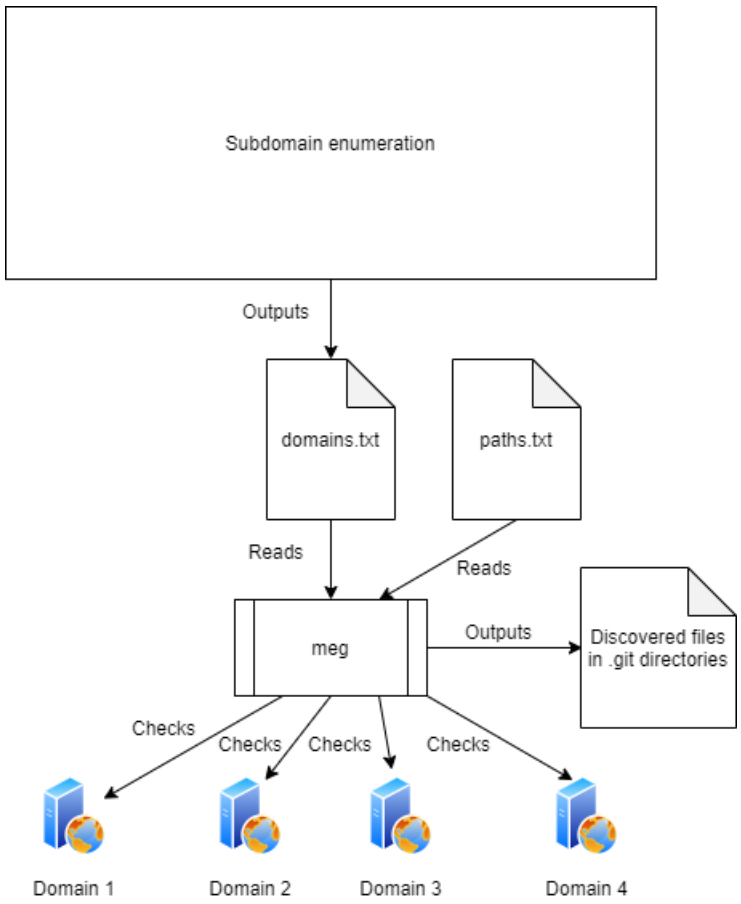

Figure 3: Workflow diagram demonstrating the .git directory discovery process

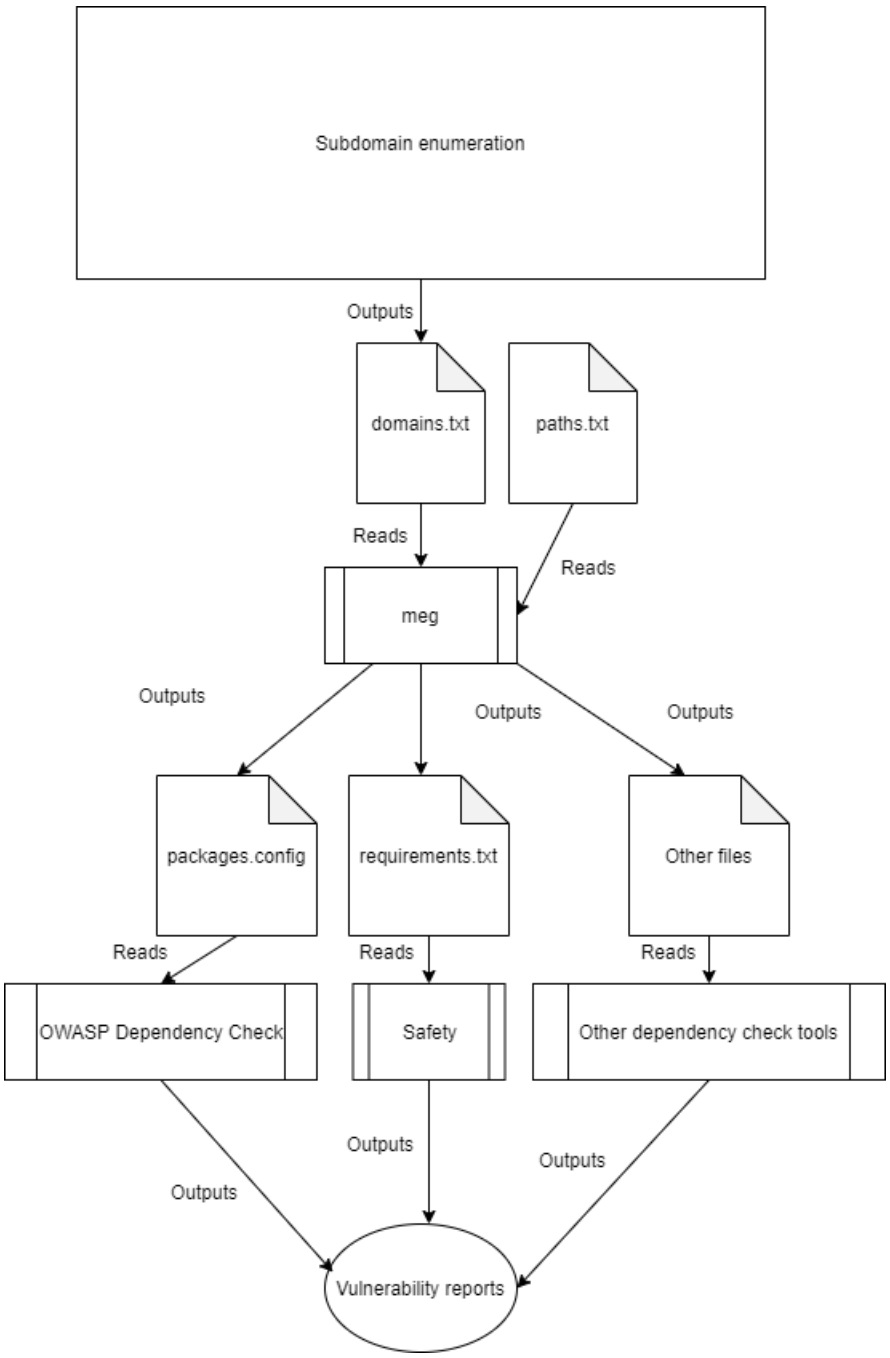

Figure 4: Workflow diagram demonstrating the vulnerability discovery process on dependency manager files

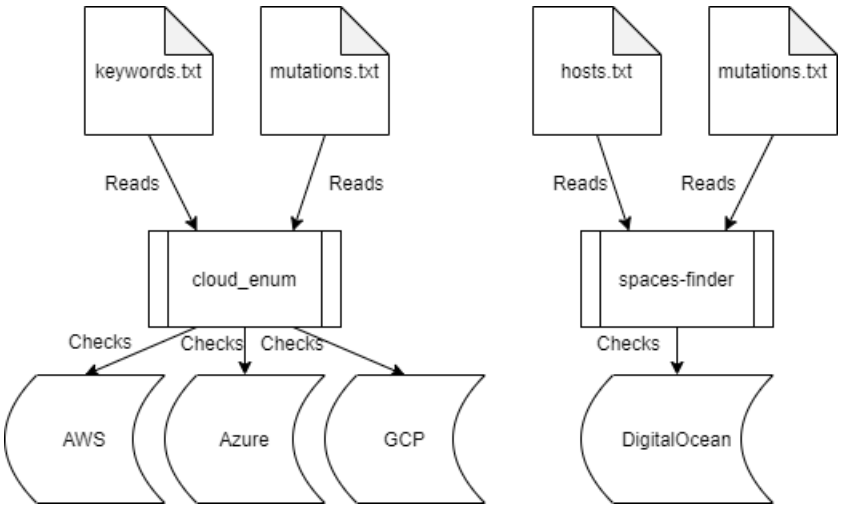

Figure 5: Workflow diagram demonstrating the discovery of misconfigured cloud storage

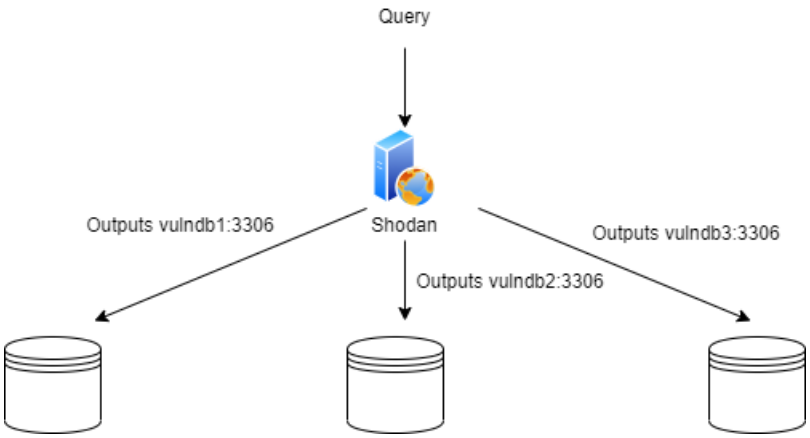

Figure 6: Workflow diagram demonstrating the discovery of databases allowing unauthenticated access

Figure 7: Keywords used to exclude responses that did not include dependency manager files

A.2 Responsible disclosure emails

```
Dear cloud provider,

During my bachelor thesis on information disclosure vulnerabilities, I
discovered a number of storage instances that potentially leak sensitive
information.
I now intend to responsibly disclose my findings to you.
Each of my requests came from either MY_IP_ADDRESS or VPS_IP_ADDRESS and
I removed all sensitive information that I came across as soon as
I discovered that it was disclosed unintentionally by a bucket owner.
Here are the buckets that I discovered:

Domains with findings

Hopefully this information is helpful to you and you can help your customers
to fix my findings. I will not share these vulnerabilities publicly
and my thesis will not contain details on the customers that were found
to be vulnerable. It will only contain the number of vulnerable domains
I found during my research and I will discuss the type of information
that I discovered to have been leaked, but not the content of any bucket.
Finally, I recognize that this is not a security problem in
cloud provider and that these types of issues are caused by errors
made by customers and I will not forget to mention this fact in my thesis.

Greetings,
Maurice Dibbets
https://www.linkedin.com/in/maurice-dibbets/
```

Listing 6: Email sent to cloud providers

```
Dear Name,

During my security research for my bachelor thesis on the top 25,000 domain
names in the Netherlands, I  discovered that your domain
vulnerable.example.com exposes a /.git/ directory at
https://vulnerable.example.com/.git/HEAD. I am now sending you
this email to inform you about the issue and I am willing
to assist you in any way possible in the process of
fixing this vulnerability.

My requests originated from MY_IP_ADDRESS and VPS_IP_ADDRESS and I did not
attempt to extract the source code of your application nor do I have any
intentions to exploit the vulnerability that I discovered, now or in the
future, regardless of your response to this email.

I hope this information is helpful to you and if you have any questions,
feel free to reach out to maurice.dibbets@student.ru.nl. I
will not share the information discussed above with anyone else, and in
my thesis I will only mention the total number of vulnerable domains that
I discovered without referring to your company in any way.

Please note that this issue is caused by a human error and that security
awareness training (as opposed to punishing your developers) is
the correct response to prevent these types of problems from occurring
in the future.

Greetings,
Maurice Dibbets
https://www.linkedin.com/in/maurice-dibbets/
```

Listing 7: Email sent to organizations exposing .git directories

```
Dear Name,

During my security research for my bachelor thesis on the top 25,000 domain
names in the Netherlands, I  discovered that your domain
vulnerable.example.com exposes a list of dependencies at LOCATION.
At least one of these dependencies has a known vulnerability in it.

I am now sending you
this email to inform you about the issue and I am willing to assist you in
any way possible in the process of fixing this vulnerability.

My requests originated from MY_IP_ADDRESS and VPS_IP_ADDRESS and I did not
attempt to extract the source code of your application nor do I have any
intentions to exploit the vulnerability that I discovered, now or in the
future, regardless of your response to this email.

I hope this information is helpful to you and if you have any questions,
feel free to reach out to maurice.dibbets@student.ru.nl. I
will not share the information discussed above with anyone else, and in
my thesis I will only mention the total number of vulnerable domains that
I discovered without referring to your company in any way.

Greetings,
Maurice Dibbets
https://www.linkedin.com/in/maurice-dibbets/
```

Listing 8: Email sent to organizations with vulnerable third-party dependencies