



Assignment of bachelor's thesis

Title:	Phishingalert.cz - forensic module
Student:	Jiří Konvičný
Supervisor:	Ing. Marek Sušický
Study program:	Informatics
Branch / specialization:	Software Engineering 2021
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

The main goal of this thesis is to extend the phishingalert.cz website with a forensic module that will identify the IP address for the sent websites, download the https certificate, detect the presence of CloudFlare or any other similar technology, scrape the entire available website, identify the used versions of libraries and modules and try to find similar ones in the history. It will also examine DNS records.

Describe similar existing projects, if any, and with their knowledge, design an appropriate project architecture. Implement, test and document the project appropriately.

Bachelor's thesis

PHISHINGALERT.CZ - FORENSIC MODULE

Jiří Konvičný

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Marek Sušický
May 16, 2024

Czech Technical University in Prague
Faculty of Information Technology

© 2024 Jiří Konvičný. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Konvičný Jiří. *Phishingalert.cz - forensic module*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

Acknowledgments	v
Declaration	vi
Abstract	vii
List of the abbreviations	viii
Introduction	1
1 Analysis	2
1.1 Phishing	2
1.2 Current state of the phishingalert.cz project	3
1.3 Requirements	4
1.3.1 Functional Requirements	4
1.3.2 Non-functional Requirements	5
1.4 Data from the website	5
1.4.1 Domain information	5
1.4.2 DNS records	6
1.4.3 SSL certificates	6
1.4.4 Used modules	7
1.5 Bot prevention	7
1.5.1 Reverse proxy	7
1.5.2 Tracking of user behavior	7
1.5.3 CAPTCHA	8
1.5.4 JavaScript fingerprint	8
1.5.5 IP address quality	9
1.5.6 Results of the bot prevention analysis	9
1.6 Ethical considerations	9
1.7 Similar projects	9
1.7.1 Reporting tools	9
1.7.2 Web analysis tools	11
1.7.3 Results of the analysis of similar projects	12
2 Design	13
2.1 Use cases	13
2.1.1 Actors	13
2.1.2 UC1: Reporting a website to <i>phishingalert.cz</i>	13
2.1.3 UC2: Viewing statistics about the reported website	14
2.1.4 UC3: Sending a report to a higher authority	14
2.1.5 UC4: Changing the application configuration	14
2.2 Application architecture	15
2.3 Server Core	16
2.3.1 Server Core Architecture	16

2.3.2	Frontend	16
2.3.3	Reporting URL to the higher authorities	17
2.4	Scraper	18
2.4.1	Visiting the reported URL	18
2.5	Database	19
2.5.1	Database entities	20
2.6	Communication between components	21
2.6.1	HTTPS	21
2.6.2	Message queue	21
2.6.3	SFTP	21
2.7	Programming language and framework	22
3	Implementation	23
3.1	Data layer	23
3.2	Server Core	24
3.2.1	Control panel	25
3.2.2	Implementation of subsequent URL reporting	25
3.3	Scraper	28
3.3.1	Obtaining domain information	28
3.3.2	Obtaining DNS records	29
3.3.3	Obtaining SSL certificates	29
3.3.4	Web crawler	29
3.3.5	Obtaining the information about used modules	31
4	Testing	32
4.1	Automated testing	32
4.2	End-to-end testing	32
4.2.1	Accessing websites with bot protection	33
4.2.2	Accessing potentially malicious websites	33
5	Conclusion	35
A	Installation guide	36
	Attachment Contents	43

List of Figures

1.1	Example of the smishing message	3
2.1	Use case diagram	15
2.2	Component diagram	16
2.3	Conceptual model of the application database	20
3.1	Screenshot of the control panel home page	26
3.2	Screenshot of the page with accident's statistics	27
4.1	Graph with the performed requests to the test page	33

List of code listings

3.1	Example of the DNS record entity	23
3.2	Example of the table object for DNS records	24
3.3	Example of the <code>IntTableRepository</code> abstract class	24
3.4	Example of part of the response from WHOIS lookup	29
3.5	Example of the method used for accessing given URL	30
3.6	Example of the web crawler configuration	31

I would hereby like to say thanks to my thesis supervisor, Ing. Marek Sušický, for his valuable advice and the opportunity to work on this project. My deepest thanks also go to my loving family, who has always supported me during the times of my studies and in my life as a whole.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 16, 2024

Abstract

This thesis deals with the analysis, design, implementation, and testing of an extension to the existing phishing reporting tool. The extension consists of two components, with the first being used to obtain data about reported websites, and the second being used to analyze and visualize these data. The developed solution can be used to get the website's domain information, DNS records, SSL certificates, used libraries and modules, and to download the raw content from the website with an automated web browser. These information are subsequently merged with those from the original phishing report, and then they can be used to report the malicious website to other subjects such as Google or Cloudflare. The final application is written in Kotlin with Spring Boot framework.

Keywords web scraping, web analysis, crawler, phishing, information security, accident reporting, Kotlin, Spring Boot

Abstrakt

Tato práce se zabývá analýzou, návrhem, implementací a testováním rozšíření pro již existující nástroj pro nahlásování phishingu. Toto rozšíření se skládá ze dvou částí, přičemž ta první z nich se stará o získávání dat z nahlášených stránek, a ta druhá tato data analyzuje a vizualizuje. Výsledné řešení zvládne získat informace o internetové doméně nahlášené stránky, DNS záznamy, SSL certifikáty, použité knihovny a moduly, a mimo to stáhne použité soubory ze stránky pomocí automatizovaného webového prohlížeče. Takto získané informace jsou poté spojeny s informacemi od toho, kdo původně nahlásil danou webovou stránku, a následně mohou být použity k zaslání hlášení dalším službám jako jsou Google nebo Cloudflare. Výsledná aplikace je napsána v jazyce Kotlin s použitím Spring Boot frameworku.

Klíčová slova web scraping, analýza webu, crawler, phishing, informační bezpečnost, hlášení incidentů, Kotlin, Spring Boot

List of the abbreviations

API	Application Programming Interface
CA	Certificate Authority
CDN	Content Delivery Network
DNS	Domain Name System
DOM	Document Object Model
DSL	Domain Specific Language
GUI	Graphical User Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OS	Operating System
PDF	Portable Document Format
PS	PostScript
SMS	Short Message Service
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
XML	Extensible Markup Language

Introduction

Phishing attacks are a threat that affects both basic users and IT professionals who try to protect them. The number of financial losses caused by these attacks has been on the rise and there does not seem to be a sign of the situation improving in the near future.[1]

One of the possible ways to combat phishing attacks is to use dedicated reporting tools in which any user can report a malicious website and hope that the responsible organization will remove the website from the Internet. These organizations consist of numerous cybersecurity and software companies, and the sheer number of their various reporting web forms adds a lot of complexity to the whole process. This can make the user feel overwhelmed, and therefore less likely to report the malicious website to the right authority.

This has been the main reason for the creation of *phishingalert.cz*, a phishing reporting tool that aims to be the primary place where Czech Internet users can report any malicious website with which they come into contact.

These malicious websites are part of the global Internet, and even though they are often hosted in high-privacy data centers, there are still useful data that can be gained from them and then used to identify the common patterns which they share with each other.

The aforementioned *phishingalert.cz* project wants to combine the information reported by the user with the forensic analysis¹ of the provided website and then forward the gained knowledge to the higher authority that has the means to act on the report. The public-facing website for phishing reporting has already been developed, and so has been a basic server-based application which is operating it. The missing parts are the module which will do the analysis of the reported website and the user interface for the project's administrator.

The primary goal is the creation of an extension to the application that provides an administrator with the ability to obtain various publicly available data about every reported website and compare the website with malicious websites from the past. These data consist of DNS records, SSL certificates, web frameworks, JavaScript libraries, and information from the WHOIS database. The related text analyzes the process of developing this extension and explains the possible ways of accessing the data, their usefulness, and the problems that could be faced during their retrieval. Some of the obtained data are also used to compare the reported website with results from the past, which can be interesting for the project's administrator, although a thorough data analysis is not part of this thesis. The data obtained with the developed tool can be used as a basis for further cybersecurity research aimed at predicting phishing attacks in the future.

The secondary goals are the development and testing of the basic administrator's user interface and rewriting of the existing server-side code in Kotlin. The administrator's user interface gives them the ability to view the data about the reported websites and re-send the reported websites with the obtained data to the higher authorities.

¹Looking for available data and finding patterns between them to determine if they are malicious or not.

Chapter 1

Analysis

This chapter introduces the reader to the issue of phishing and explains the related concepts. It shows the current state of the *phishingalert.cz* project before the writing of this thesis and describes in detail the concrete requirements which were agreed upon with the thesis supervisor. This is followed by a review of similar existing solutions and the analysis of mechanisms which are used to prevent the automated tools from accessing the web.

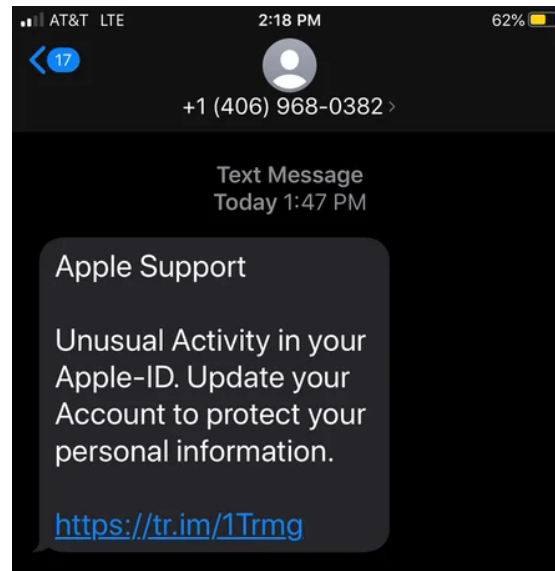
1.1 Phishing

Phishing is a type of cybercrime that focuses on the acquisition of sensitive data from the user that can be used for financial or identity theft. The perpetrator often tries to get access to data such as the victim's username, password, account number, home address, and credit card details. This is done by masquerading as a legitimate institution or a personal contact that the victim already trusts. [2]

The cyber security community recognizes several types of phishing attacks according to their potential victims and the methods used to carry out them. Intended users of the tool developed in this thesis may potentially encounter these types of phishing attacks:

- **Smishing:** The name of this phishing type comes from the combination of “SMS” and “phishing”, and as the name implies, it uses messages sent through SMS or instant messaging services to distribute malicious URL to the victim. The messages can be divided into two types. The first type looks as a message from a trusted and reliable source such as the victim's banker or system administrator. The second type of smishing tries to abuse the victim's response to a shock by informing them about a serious event, such as their identity or account being stolen, and scaring them with possible consequences which would occur if they do not respond. [3]
- **Vishing:** Unlike smishing, this phishing type uses voice calls to put pressure on their victim and lure sensitive personal or corporate information from them. [2] Vishing capitalizes on the fact that their typical phishing target usually puts more confidence in the safety of telephone service and is not aware of possible scamming techniques such as caller ID spoofing. The development of the IP telephony system has made the whole vishing process easier because the perpetrator is not limited only to the classic mobile network carriers. [3]
- **Social media phishing:** Social media are also popular among the perpetrators who tend to utilize mainly the messaging capabilities of the platforms, for example Facebook Messenger or Twitter DMs. [4] There has also been a growing number of phishing links that were delivered

as promoted posts or paid advertisements on the user's social media feed. The companies behind these platforms were reported to not act adequately in some cases. [5]



■ **Figure 1.1** Example of the smishing message targeting Apple users [6]

1.2 Current state of the phishingalert.cz project

This section familiarizes the reader with the state of the *phishingalert.cz* project before the implementation of the features that were agreed upon with the supervisor of this thesis.

The project was originally developed by Ing. Marek Sušický as a non-profit tool that can be used by anyone who wants to report a phishing website on one place without having to manually find the competent authority who is responsible for deleting the malicious website. The project itself currently consists of two components:

- **Frontend:** The frontend is represented by one simple web page with information about the whole reporting process, which also contains a form for filling the details about the encountered phishing attack. The website is created using HTML and JavaScript, which is used mainly for input validation. It is hosted on Github Pages¹ and runs on a different server than the backend, with which it communicates through HTTP POST requests. Other operations than sending a phishing report to the backend are not supported.
- **Backend:** The backend is written in Python with Flask² framework. It provides an API which supports submitting a phishing report and confirming it through the link which is sent to the email of the report's author.

The information about the phishing report and its author is saved in the database which consists of a single table. In addition to the information provided by the author of the report, the stored data also include information about the author's IP address, User-Agent of their browser, and whether the report is confirmed or not.

As the description of the existing components implies, the number of currently supported features is not large, and there is room for major potential improvements.

¹<https://pages.github.com/>

²<https://flask.palletsprojects.com/en/3.0.x/>

1.3 Requirements

The final application must meet the requirements that were agreed upon with the supervisor of this thesis. Functional requirements specify concrete functions and features of the product, and non-functional requirements describe the more general properties and user experience of the final product. [7]

1.3.1 Functional Requirements

1.3.1.1 F1: Storing of website information

The application should store information about each reported website in persistent storage. The stored information should contain at least the following entries:

- IP address
- Domain registrar
- Date of domain registration
- DNS records
- SSL (HTTPS) certificate if present
- Website's source code and related files
- Used modules (libraries and frameworks)
- User-written note

1.3.1.2 F2: Filtering of phishing reports

The administrator can filter stored reports using common patterns, such as the email of the person who reported them.

1.3.1.3 F3: Configuration

The application can be configured using a human-readable configuration file. The configuration should be easily extensible in the future.

1.3.1.4 F4: Bypassing defense against web scraping

The application should implement some techniques to combat anti-scraping measures which could be used by reported websites. It should bypass at least the basic Cloudflare anti-bot protection and avoid triggering CAPTCHAs.

1.3.1.5 F5: Reporting of received websites

The received websites will be reported to the appropriate authorities consisting of cybersecurity organizations and tech firms. These organizations will be chosen from the pool of registered organizations in the control panel. The whole process should be automated as much as possible, although the final decision to send these reports should be made by the administrator.

1.3.1.6 F6: Finding similar websites from the past

There should be the possibility to return a list of similar websites. The list will be sorted by an algorithm that takes into account some of the prominent patterns on those websites.

1.3.1.7 F7: Control panel for the administrator

The administrator should be able to view the data from the reported websites in a control panel with a graphical user interface. This control panel should be protected with password authentication to mitigate the access of unauthorized users.

1.3.2 Non-functional Requirements

1.3.2.1 N1: Accessibility of the control panel

The control panel for an administrator should be implemented as a website that will be accessed from a desktop computer. It should support at least Google Chrome and Mozilla Firefox web browsers.

1.3.2.2 N2: Open source code

The application source code should be public and published under an open source license.

1.3.2.3 N3: Installation guide

The final product should have a clearly written installation guide that could be used by anyone with the required dependencies to test the developed product.

1.4 Data from the website

This section discusses the specific data from the website that will be obtained by the developed application and potentially analyzed by it.

1.4.1 Domain information

Every website has a domain name, which is a string of text that is further resolved to an alphanumeric IP address which is used when accessing the resource on the server. [8]

Each domain is managed by a *domain registry*, which delegates the reservation of domain name to *registrars*, who handle the registration process with the entity that wants to register the particular domain. Everyone who reserves a top-level domain name³ must fill out their personal information in the “WHOIS database”.

This “database” is actually a set of independent databases maintained by selected domain *registrars*, which can be queried using the WHOIS protocol. The protocol returns information about the given domain in a human-readable format. [9] The response usually includes *registrant's*⁴ and *registrar's* name and contact information, the registration date, the expiration date, and the name servers. [10] The information in the response can vary depending on the top-level domain, and the response format is not completely standardized. [11]

Due to issues such as lack of standardization of the WHOIS response, a new protocol called RDAP⁵ was developed, which returns the registration data in a structure that can be easily read

³E.g. registering domain names such as google.com or cvut.cz

⁴Owner of the domain

⁵Registration Data Access Protocol

by a computer program. This protocol should eventually replace WHOIS, but currently both are used in parallel. [11]

Registrars usually provide an option of private registration, in that case *the registrar's* personal information are filled in the “WHOIS database” instead of the user's. The actual user's information is held in *the registrar's* private database, and its security depends on the credibility of *registrar*. [12]

This means that the developed program will not have direct access to the personal information of a malicious website's owner because they will probably use *a registrar* providing a private domain registration. However, the developed service can potentially use public data about *the registrar* to contact them and coordinate further proceedings which could lead to the removal of the malicious website.

1.4.2 DNS records

DNS record, also known as a zone file, is a set of instructions that are used to connect the domain name to the corresponding IP address within the DNS server. [13]

In simplified terms, the information encoded in each DNS record consists of the following data, some of them being optional [14, pp. 10–20]:

- **Domain name**
- **TTL:** Marks the time for which a record may be temporarily stored in the local cache, this field is optional.
- **Class:** A DNS record can theoretically be in one of the multiple classes, in practice only the Internet class (IN) is used, and this field is optional.
- **Type:** DNS record type helps determine the kind of information stored inside the record. Some of the common DNS record types are *A* (marks an IPv4 address corresponding to the given domain), *AAAA* (similar to *A*, but the address is an IPv6), *MX* (marks a server that is responsible for a mail exchange coming to the given **Domain**), and *NS* (marks an authoritative name server for the given **Domain**).
- **Preference:** The preference is present only in some DNS record types, i.e. in *MX* record. Records with lower values are preferred over those with higher values.
- **Record data:** This field contains the data to which the domain name is resolved. The data are of the aforementioned **Type**.

These records can be useful for spotting various network-related patterns on reported websites. It can be analyzed how many of them use IPv6, which websites also have a dedicated mail server, or which name servers are popular or shared among them. If the website uses a name server from some particular company like Cloudflare, there is also a possibility that they will use their anti-bot protection or other related services which the company provides.

1.4.3 SSL certificates

The SSL certificate, also colloquially referred to as the HTTPS certificate, is a variant of the X.509 cryptographic certificate used with the SSL protocol. [15] This protocol is used as part of the HTTPS, and by reading the server's SSL certificate, the client can verify the identity of the server which hosts the website or different requested resource. [16]

This certificate's structure consists of multiple fields which provide details about the entity to which it was issued and other associated data. The structure's fields can be simplified into the following points [15]:

- **Serial number:** A unique integer given to the certificate by the CA (Certificate Authority). It is unique between all certificates issued by the given CA, but not globally unique.
- **Signature algorithm:** The cryptographic algorithm which was used to sign the certificate by the CA.
- **Signature:** Digital signature computed from the part of certificate with the Signature algorithm.
- **Issuer:** Contains the information about the CA which signed and issued the certificate.
- **Subject:** Identifies the entity to which the certificate was issued and which owns the Public key.
- **Validity:** Specifies the date from which the validity period of the certificate begins, and the date when it ends.
- **Public key:** Used by client for verifying the identity of the server.

1.4.4 Used modules

This general term includes all external dependencies of the website. Some of the examples are JavaScript libraries, frameworks, content management systems, ad networks, or used fonts. All these modules can be potentially described by their **name**, **version**, and **category**.

Information about the used modules can be further utilized to group the websites into categories according to their dependencies and compare the similarity of their technological background.

1.5 Bot prevention

The analysis of the reported websites is performed with automated tools (bots) that potentially face the defense mechanisms mentioned below. However, these mechanisms can also have a negative impact on legitimate users. This can motivate the teams behind anti-bot services to not fully utilize such mechanism, and in that case it is mentioned in the related paragraphs.

1.5.1 Reverse proxy

Reverse proxy is an application which is placed in front of selected web servers and intercepts requests from clients trying to access the web servers behind it. It is used for caching, as a solution against DDoS⁶ attacks, and also for blocking suspicious requests that do not seem to come from a normal user. [17]

The reverse proxy itself does not have to provide any kind of anti-bot protection, but when it does, it can use some of the techniques which are further explained in the following subsections.

1.5.2 Tracking of user behavior

A detection module for an anti-bot service can utilize the fact that a human user behaves differently than a bot controlled by a deterministic algorithm.

The behavior patterns can be potentially divided into the following points [18]:

⁶Distributed Denial-of-Service

- **Maximum sustained click rate:** A human accessing the website is physically limited to perform only a finite number of mouse clicks, and therefore only a finite number of corresponding HTTP requests. This means that there is a threshold that is very unlikely to be exceeded by someone who is not a computer program.
- **Duration of session:** When someone visits the website, the unit of time between the first and the last request is usually shorter if it is a human than if it is a bot. This comes from the observation that the browsing behavior of a human is more goal oriented and focused than the behavior of a bot.
- **Percentage of image requests:** The share of image requests (e. g. *.jpg* and *.gif* files) in the total number of requests tends to be higher for a human than for a web crawling bot, which is usually interested in text and HTML resources⁷. [19, p. 896]
- **Percentage of PDF/PS requests:** In contrast to previous point, some web crawling bots tend to make requests to view PDF and PS files more often than humans. [19, p. 896]
- **Percentage of HTTP 4xx responses (errors):** Compared to humans, bots have a higher percentage of HTTP error codes in responses to their HTTP requests.
- **Request for *robots.txt* file:** Another thing that can be checked is the access of the client to the file called *robots.txt*. This file contains a set of instructions for web crawling bots that informs them about the sites which then can and cannot access. [20] Some bots tend to ignore these instructions and do not make a request to access this file, which is not something that a legitimate user would intentionally do, and therefore it raises suspicion.

1.5.3 CAPTCHA

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a security measure which is used to block bots from accessing the protected resource on a website. It is implemented as a simple task which is easy to do for a human, but complicated for a computer program. [21]

This test is generally seen as an effective method to prevent bots from accessing the protected resource, but its usage is not popular among users because it slows them down in their work. The difficulty of CAPTCHAs has also increased over the years because of machine learning algorithms that can break through older versions of it. Still, some CAPTCHA bypassing is possible due to specialized companies which provide the option to solve the tests by their employees from third world countries for an affordable price. [22]

1.5.4 JavaScript fingerprint

This broad term includes all types of anti-bot measures that depend on execution of JavaScript code. It should be noted that the usage of these types of test can be controversial, because it penalizes legitimate users who block the execution of JavaScript in their browsers due to performance or privacy concerns, and therefore it is not a perfectly reliable bot detection mechanism. The authors of bot detection algorithms seem to be aware of it and most of the anti-bot services use these kinds of test just as a part of the bot detection mechanism, not fully depending on it. [23, p. 148]

However, a large amount of information about the connected device can be gathered in this way. This includes information about the browser, the operating system, and the hardware, which can be used to classify possible bots. [23, p. 138]

⁷This obviously does not apply to all bots, as some of them may be actually interested in images even more than in the text content.

1.5.5 IP address quality

The source of the IP address can also be used as one of the factors when the protected server decides if the incoming request is legitimate or not. Most of the bot traffic comes from the range of IP addresses which is assigned to public clouds such as AWS⁸ and Microsoft Azure⁹. [24] Some anti-bot services therefore block the requests from such IP addresses or treat them as more suspicious, although the overall sensitivity of the service to the public clouds highly depends on the concrete implementation. [23, p. 156]

Creators of malicious bots may try to avoid detection by hiding in residential traffic with IP addresses that are owned by someone else. Fighting this approach is much harder because it can lead to an unwanted punishment of legitimate users, who can get completely blocked or trapped in “*the CAPTCHA hell*”, which means they have to solve the CAPTCHA every time they try to access the protected website. [24]

1.5.6 Results of the bot prevention analysis

After going through the list of possible defense mechanisms that can prevent automated tools from accessing a website, I decided to keep some of them in mind during the development of the final application.

These are the maximum sustained click rate (the developed tool will visit the URLs with a limited speed to try to avoid it), percentage of image requests (it will load them in the same way as the real user would do), request for *robots.txt* file, and JavaScript execution.

1.6 Ethical considerations

It would be ideal if the developed tool visited only malicious websites while completely mitigating any kind of access to the harmless part of the Web. However, any solution that implements this approach would be less effective because it cannot be clearly concluded which URL is safe and which is not without manually checking it first, which is unfeasible for an automated tool. This means that there will be at least a small number of harmless websites that will be visited and analyzed, even though they do not have to.

To mitigate unnecessary strain on server resources, the developed tool will have the option to visit each URL only once in a given time frame, whose length will be configurable.

1.7 Similar projects

As of February 2024, there are multiple existing projects that combine the reporting of malicious websites with their analysis and, therefore, are relatively similar to the *phishingalert.cz* project.

Given the assignment of this thesis, I have decided to go into more detail and split this section into two parts. The first part describes existing tools for malicious websites reporting, and the second part shows the projects which are focused solely on the website analysis.

1.7.1 Reporting tools

There are multiple tools that the user can use to report a malicious URL or website, and although the user experience may be similar, their approach to handling the reported data can be quite different based on the authority behind it.

⁸<https://aws.amazon.com/>

⁹<https://azure.microsoft.com/en-us/>

1.7.1.1 URLhaus

This service, which started as a research project at the Bern University of Applied Sciences [25], is used to share malicious URLs with the aim of eliminating them by collaboration with various antivirus vendors and blacklist providers. [26]

The whole project uses reports from registered users, who can also add tags with additional information (malware type, malware's CPU instruction set, etc.) or make a request to mark the reported URL as a false positive. The report can be made through the web GUI or through the public API. [27]

Although the general use case of URLhaus is the reporting of malware hosted at the given URL and not the reporting of phishing websites, its philosophy of sharing gained data with third parties is similar to the approach of the *phishingalert.cz* project. Based on the statistics published on the URLhaus website, 72.5 % of the reported URLs have been blocked by Cloudflare DNS [28], although it is not clear whether it is the result of a combined effort of multiple parties or just URLhaus alone.

The further analysis of the malicious URL payload is done through external services such as VirusTotal and MalwareBazaar. [29]

1.7.1.2 PhishTank

Phishtank is a community-based site for phishing reporting operated by Cisco Talos Intelligence Group. Users can submit phishing URLs and vote for existing submissions to confirm that they consider the submitted URL to be malicious. [30]

Anyone with a registered account can send the malicious URL to PhishTank via their website or send it to `phish@phishtank.com` from the previously registered email address. There is also a public API for getting information about given URL in XML or JSON format, the response then contains data such as submission date, validity, and whether the reported URL is confirmed to be malicious. [31] It is also possible to download the whole PhishTank database, which is useful for a service that will perform many lookups into the data. [32] The usage of PhishTank is currently limited due to the closed registration of new accounts, although the already registered users are still able to add new reports. [33]

1.7.1.3 Google Safe Browsing & Web Risk

The tech company Google, known for its search engine and cloud computing services, also has its own phishing reporting solution, which is split into two parts: *Safe Browsing* for common users and non-commercial applications, and *Web Risk* for commercial solutions. [34]

The main public-facing part of *Safe Browsing* service is a web form, which consists of two basic fields, one for the malicious URL and one for the additional details. The form is protected by reCAPTCHA¹⁰ to combat possible spam. [35]

Both services provide an API which can be used to look into Google's list of unsafe web resources. This list includes both phishing URLs and malware URLs. [36] The *Web Risk* API also includes the option to submit an URL with malicious content that is further reviewed by Google, although only 100 submissions per month are allowed in the free tier of the *Web Risk* service. [37]

Google uses gained data about malicious websites to warn users who try to access the unsecure website from their list. This is usually done by redirecting the user to a dedicated warning website that informs them about possible risks. [38]

¹⁰<https://www.google.com/recaptcha/about/>

1.7.1.4 Reporting to Cloudflare

Cloudflare is a CDN provider that also has its own reporting web form hosted at abuse.cloudflare.com. The form contains fields for author contact info, evidence URLs, logs, and optional comment. [39]

Cloudflare explicitly states that it usually cannot remove the reported website from the Internet because it actually hosts only a handful of reported websites. Its services are used mainly as a CDN or a pass-through security node that sits in the middle of the Internet traffic. In that case, Cloudflare at least forwards the complaint to the website's hosting provider, and the further action depends on their decision. [40]

1.7.2 Web analysis tools

The projects listed here were analyzed to see the state of existing solutions which are similar to the forensic module being developed as part of this thesis. They were also inspected to get an idea of all the data that can be gathered from the analyzed websites and the possible ways to retrieve them.

1.7.2.1 urlscan.io

This service describes itself as “*a sandbox for the web*” [41] and its main functionalities are scanning and analyzing the requested URLs. The whole process starts by visiting the given URL with a headless browser masquerading itself as a regular user. The URL can be accessed from one of the 22 countries provided, and the browser's User-Agent can be customized to better mimic the real user's behavior. The notable data in the result consist of a list of IP addresses that were contacted during the page load, executed HTTP requests, SSL certificates, the size of the transferred data, DOM content, used web technologies, JavaScript global variables, and created cookies. [42]

However, it does not support the downloading of the JavaScript source code, CSS, and all available DNS records.

There are two ways to request a scan of an URL: one through the web user interface and one through the API, which can be used for free with a limited rate of requests.

The scan results are divided into three categories based on their visibility. Public scans can be viewed by anyone on the *urlscan.io* front page and in the search results. Unlisted scans can be accessed by the subscribers of *urlscan Pro* platform, and private scans are visible only to those who know its unique ID. [43]

1.7.2.2 Cloudflare URL Scanner

The URL Scanner from Cloudflare [44] is relatively similar to the aforementioned *urlscan.io* service. It lets the user set up a custom User-Agent, HTTP referrer¹¹, or any other HTTP header that is then used in the request, which is performed by an automated Google Chrome browser controlled by the Puppeteer¹² library. [45]

The URL Scanner can take a screenshot from multiple targets, such as a smartphone or a desktop. It also logs all HTTP requests made between the browser and the requested website. The scanned data include the DOM content of the page, the JavaScript global variables, cookies, the HTTP version, and some of the technologies used by the page.

Based on scan requests for a few websites, the list of identified technologies is sometimes shorter than in the case of *urlscan.io*, and the DNS records are also incomplete, although at least the A records seem to be present.

¹¹HTTP header which specifies the URL of the web page from which the request was made.

¹²<https://pptr.dev/>

The website scan can be requested from the web user interface or through the URL Scanner API, and the visibility of the scan can be either public or unlisted.

1.7.2.3 Wappalyzer

Unlike *urlscan.io* and *Cloudflare URL Scanner*, this tool focuses on the concrete technology used by the website and does not attempt to show any statistics on the performed HTTP requests or other metrics associated with Internet traffic. It has its own database of web-related technologies, such as content management systems, frameworks, JavaScript libraries, analytics services, or CDNs. [46]

Their detection system finds the used software by inspecting website's source code, JavaScript variables, HTTP headers, or cookies. There, it looks at information leaked by the used technology and compares it with the records from their database. [47] The lookup can be performed using the Wappalyzer browser extension, API, or one of the supported CRM¹³ systems. [46]

The downside of this service is its business model. The cheapest API option starts at \$250 per month [48], and even though the browser extension is free, its usage in an external project is obviously very limited.

1.7.3 Results of the analysis of similar projects

After going through existing projects focused on reporting of malicious websites and their analysis, I conclude that none of them fully comply with the assignment of this thesis. However, several observations have been made that may help in the design of this project.

The first is the idea of saving the screenshot of the reported page in a way similar to *urlscan.io*. This approach seems useful because it can give the project's administrator an insight about the reported page without the need of visiting it, and it can also serve as a backup in case the downloading of the page's source code fails and the original page gets deleted.

The second observation is related to the sharing aspect of some of the mentioned projects. Reporting phishing accidents to the higher authorities was already suggested before by the supervisor of this thesis, but the review of existing solutions gave me a more concrete idea of a possible approach to do this. For example, the Cloudflare reporting tool states that it sends the submitted report to the hosting provider of the reported website [40], which can be utilized to reach those hosting providers with relatively low effort produced, although it is obviously still less effective than getting to them directly. Knowledge of *Google Web Risk API* also inspired me to support it as a functionality in the developed project.

The analysis has also introduced me to the public sources of malicious websites, the sources being URLhaus and PhishTank. The data from their databases can be used to test the project with real-world data later on and assess the reliability of the developed solution.

¹³Customer Relationship Management

Chapter 2

Design

This chapter describes the requested use cases of the project, the architecture and components of the developed application, the design of required functionalities, and the technologies used.

2.1 Use cases

This section describes the actors who will use the application and possible tasks they can perform, thus creating use cases of the application, which are also visualized in the included use case diagram.

2.1.1 Actors

There are two actors who are relevant to the developed application:

- **Administrator:** The administrator is a privileged user who can view data from all reported websites and change the configuration of the application. They have a background in cybersecurity and have at least intermediate knowledge of web applications and related technology.
- **User:** The user is a person who reports suspicious websites in the web form. It can be a person from the general public without any significant IT skills. They provide their own contact information for possible future interactions with the administrator and to verify that they are not a bot.

2.1.2 UC1: Reporting a website to *phishingalert.cz*

Actor: User

1. User opens the website with reporting form
2. User enters their contact information and information about the reported website into the web form, this creates a new accident report
3. System checks validity of the entered data and prompts the User to edit it if it contains errors
4. System saves the accident and sends confirmation link to the user's email address
5. User opens their email client and clicks on the link
6. System changes the status of the reported accident to confirmed

2.1.3 UC2: Viewing statistics about the reported website

Actor: Administrator

1. Administrator opens the website with control panel
2. System shows recently added reports
3. Administrator finds the report for which they were looking for and clicks on a *Statistics* button
4. System shows statistics of the report

2.1.4 UC3: Sending a report to a higher authority

Actor: Administrator

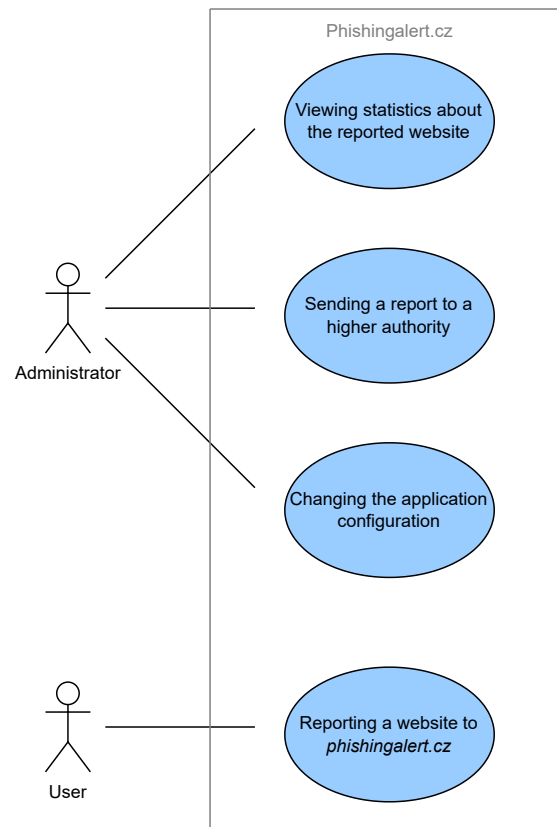
1. Administrator opens the website with control panel
2. System shows recently added reports
3. Administrator finds the report for which they were looking for and clicks on a *Report website* button
4. System shows a drop down menu with available authorities
5. Administrator clicks on the desired authority
6. System performs the reporting process

2.1.5 UC4: Changing the application configuration

Actor: Administrator

This use case is partially done outside of the application.

1. Administrator goes into the working directory of the application and opens the configuration file which is located there
2. Administrator changes the required parameters in the configuration file, saves the file and restarts the application
3. Application starts correctly if the configuration is valid, otherwise it informs the administrator about the issue, which can be fixed by repeating the previous step

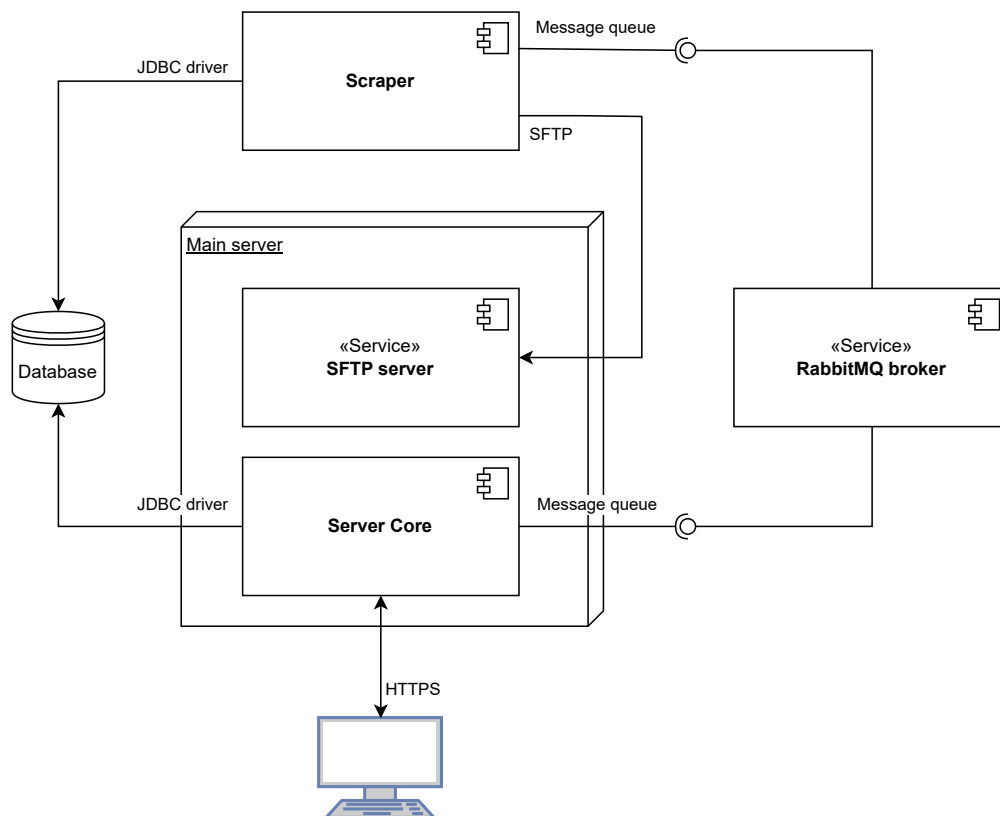


■ **Figure 2.1** Use case diagram

2.2 Application architecture

The application architecture was designed in a way that respects the core software engineering principles of low coupling and high cohesion.

It consists of three supporting services (*Database*, *RabbitMQ broker*, *SFTP client*) and two main components (*Server Core* and *Scraper*) whose concrete implementations are independent of each other.



■ **Figure 2.2** Component diagram

2.3 Server Core

Server Core is the heart of the entire application that is exposed to the Internet. It handles all incoming requests and contains business operations such as user authentication and website analysis.

This component implements the logic behind the entire reporting process, and it also controls the rendering of the frontend content that is subsequently served to the user or administrator.

2.3.1 Server Core Architecture

The implementation of the Server Core component is based on the MVC pattern. This pattern consists of three main elements: the Model, the View, and the Controller.

The Model is a layer which is responsible for defining the entities in the application and the operations around them, such as their storing and retrieving. *The View* layer handles the presentation of the data to the user and *the Controller* contains the logic for handling input from the user and updating *the View* or *Model* according to it. [49]

2.3.2 Frontend

The GUI of the application is implemented as a web frontend with a public and an internal part. The public part consists of a web form which is used for filling in the information about a phishing attack. The internal part, to which is also referred to as the “control panel”, is used by the administrator of the whole system.

2.3.3 Reporting URL to the higher authorities

One of the most needed features was automating the process of reporting URLs to the higher authorities. Before developing this thesis, the project administrator needed to manually copy the information about the reported websites from the database and paste it into the reporting tool of the authority they deemed appropriate.

There are multiple ways to do the reporting process programmatically, each of them having its pros and cons.

- **API:** Using the public API from the authority itself is the easiest and most reliable way to send the phishing report. The structure of inserted data and the possible responses from the server are both well known, which makes the reporting process deterministic and less prone to bugs.

The reporting process can be **partially** or **fully automated**. The fully automated process would automatically submit a reported URL to an API, while the partially automated process would do this after the administrator's action.

The problem lies in the fact that only part of the relevant authorities provide this option and if they do, there are still other limitations, such as rate limiting, which is the case of Google Web Risk analyzed in 1.7.1.3.

- **Fully automated without API:** From the administrator's high-level perspective, this kind of approach might feel very similar to the usage of an API – just click the button / change a setting in the configuration file and let the system handle the rest.

The problem lies in the potential implementation of this approach. Because an API is not present, the whole process would need to be done by some kind of bot which would fill out the reporting web form and which would potentially need to pass the protections mentioned in the section 1.5. The most problematic would be the passage through CAPTCHA. Based on my observations of multiple reporting tools, this obstacle is almost always present in them.

Then there is an ethical problem with breaking the Terms of Service of the used web form because the firms behind them do not assume that they will be filled by a bot. For example, Google's Terms of Service state that the user cannot participate in “*bypassing our systems or protective measures.*” [50]

There are also potential problems in the future if the reporting form changes its layout or implementation. An automated tool would not be able to cope with these changes and could cause some serious bugs such as submitting incorrect or incomplete data. This approach could be potentially harmful and it could defeat the whole purpose of the *phishingalert.cz* project.

- **Partially automated without API:** This approach combines the concept mentioned above with a human interaction. The data is automatically inserted into the reporting web form by the program, while the administrator has to solve CAPTCHA and confirm the sending of report to the authority. The administrator sees the inserted data before sending them, making the whole process safer and less prone to accidental spam. It does not violate the usual Terms of Service agreement because the report is technically being filled by a human.

The only downside is the consumption of the administrator's time, although it is still significantly faster than the old process, which was completely manual.

After going through possible URL reporting options, it was decided to implement two reporting strategies – one with **API**, and one without it. Both options are implemented as a **partially automated** solution. Fully automated solutions could face problems with rate limiting from the server or sending wrong URLs. This does not mean that it could not be reliably done, but the difficulty of such a process goes beyond the scope of this thesis.

2.4 Scraper

Scraper is a standalone component that is focused on finding and downloading the required information about the website. It takes care of the whole download process and communicates with the Server Core through a message queue.

The main reason for the choice of the independent Scraper component is the possible need to run the whole scraping process from a machine different from the main server. It also enables parallel usage of various Scrapers which can handle multiple incoming requests from the message queue without waiting until the currently busy Scraper finishes.

2.4.1 Visiting the reported URL

Some of the data associated with the website must be accessed by visiting the reported URL with the related pages and downloading the content directly from them. This can generally be done with a web crawler, which is a piece of software that can systematically browse the web and access the requested data.

I have established several criteria that should be met by the web crawler used in this project:

- 1. URL filtering:** The crawler should support at least a basic filtering of the URLs referenced from the visited website. This is important because some phishing websites contain URLs to the legitimate page of the organization that they try to impersonate, and visiting these secure URLs would be a waste of time and computing power. It could also put unnecessary strain on the organization's server infrastructure, which is not desirable.
- 2. Human-like behavior:** The crawler should try to imitate a real user who visits the website, because some refined phishing websites could serve different content to the automated visitor or use the bot prevention mentioned in 1.5.
- 3. JavaScript execution support:** The execution of JavaScript code is a must due to the fact that the reported website can be written in a framework like React¹ or Vue.js² that is highly dependent on it. Crawlers without JavaScript support are also more likely to be blocked by anti-bot protection. [23, p. 149]
- 4. Modifiable code:** It should be an open source software with permissive license and reasonable documentation that can be modified for the needs of this project. This means that it should be ideally written in a JVM language as the rest of the project.
- 5. Actively maintained:** The web crawler should be actively supported and maintained. The project is considered maintained if the last commit in its source repository is not older than 6 months.

With the aforementioned criteria in mind, the process of finding the appropriate web crawler has started. Criteria 1 and 3 alone discarded a large number of web crawlers found [51], and after applying the remaining points, only two options remained in the selection process, the options being *Apache Nutch*³ and *Apache StormCrawler*⁴.

Apache StormCrawler is designed to be used with the *Apache Storm* distributed computation system and therefore was discarded. The remaining *Apache Nutch* initially seemed like a good choice, but it would have to be heavily tuned to adhere to the criteria number 2. Given the size and complexity of the existing Apache Nutch project, this was evaluated as a too time-consuming task with uncertain outcome.

¹<https://react.dev/>

²<https://vuejs.org/>

³<https://nutch.apache.org/>

⁴<https://stormcrawler.apache.org/>

Therefore, I have decided to write my own web crawler without relying on existing solutions. The developed web crawler is based on the automated browser, which is controlled by the Playwright library.

This solution trivially supports the criteria number 3, because the JavaScript support is integrated into any modern web browser and the criteria numbers 4 and 5 are also satisfied, since the web crawler is developed as part of this thesis. The parts that need to be programmed are URL filtering and human-like behavior based on the observations from Results of the bot prevention analysis.

The Playwright library, which is primarily focused on end-to-end testing of web applications, can be used with all major browser rendering engines including those used by Firefox and Chromium. Its development is backed by Microsoft and the library offers features such as automatically waiting for required page elements to load or emulating mobile devices. [52]

I have chosen Mozilla Firefox as the main browser for the web crawling task because it seems less susceptible to being detected by anti-bot protection than Chromium, although not by a huge margin. [23, p. 148]

2.5 Database

The Database handles saving and reading of the data in a structure that enables fast execution of queries and effective storage space utilization.

Nowadays, there are two main types of databases that can be described as follows [53]:

- **Relational databases:** These databases store the data in tables with rows and columns, with each column representing a specific data attribute and each row representing an instance of that data. Every table has a primary key that can be used to specify the relationship between data in the tables, and the user writes SQL queries to interact with the database. This kind of scheme is useful for use cases where the relationship between entities is important. Relational databases follow strict ACID⁵ properties, which means that the database is able to maintain data integrity despite errors in data processing, and the data should be accurate at all times.

When it comes to performance, these databases are usually vertically scaled by adding more CPU cores and RAM to the server. Horizontal scaling is possible by duplicating data across multiple servers for read-only workloads, but it is not trivial.

- **Non-relational databases:** This type of database, also called **NoSQL databases**, uses a less strict and more flexible scheme than a relational one. Data can be stored as a collection of key-value pairs, JSON objects, or graphs with nodes and edges. This approach can be useful for data that are flexible in shape or size and may change in the future.

Non-relational databases are not usually compliant with ACID because most of them are only eventually consistent, which means that the queries can return old data for some period of time. [54]

These databases can be horizontally scaled by adding more nodes (servers), and the whole process is more straightforward compared to the scaling of relational databases.

After going through the pros and cons of each database type, it was decided to use a relational database. It is highly unlikely that the number of accessed data would exceed the performance limits of the relational database in the future, and the rigid scheme is suited to the nature of stored data whose structure will not change very often.

PostgreSQL⁶ was chosen as a relational database implementation because it is open source, widely adopted, and the project has already used it.

⁵Atomicity, Consistency, Isolation and Durability

⁶<https://www.postgresql.org/>

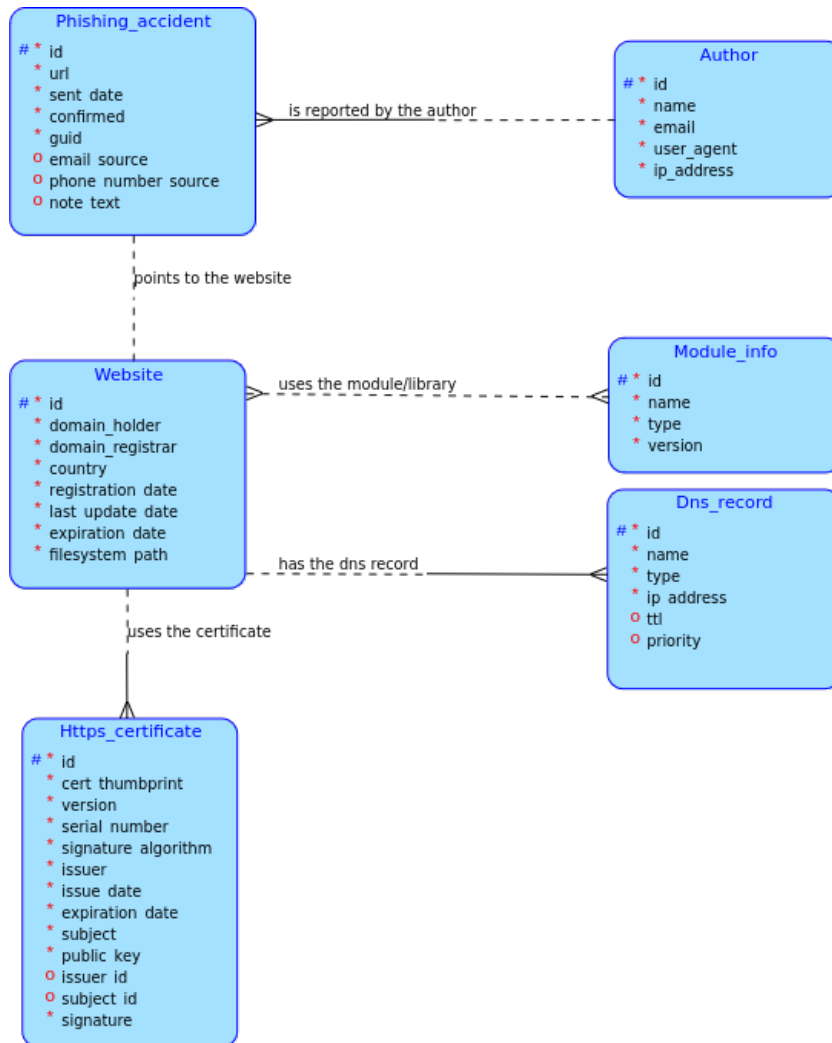
2.5.1 Database entities

The entities in the database are represented and connected in a way described in the conceptual model of the application database in the figure 2.3.

The Author and Phishing accident are filled in by *the Server Core*, while the rest of the data is provided by *the Scraper*. There is a possibility that the user does not send the confirmation of their report (Phishing accident), which is part of the process described in 2.1.2, and scraping does not occur. Due to this, the relationship between the data from *the Server Code* and the data from *the Scraper* is only optional.

The Phishing accident and the Website entities are divided into two separate parts for the same reason. They could theoretically be put into a single table, but then the Website's rows would need to be nullable, which initially seemed as a worse option, although the chosen solution also has its pitfall in the extra step that is needed in database queries.

The attributes of the entities are based on the previous analysis performed in the Data from the website section.



■ **Figure 2.3** Conceptual model of the application database

2.6 Communication between components

Since the application is divided into multiple components, it is necessary to find a way for them to communicate with each other. When it comes to implementing a communication protocol, there are two approaches that can be taken [55]:

- **Synchronous protocols:** Synchronous protocols require the sender and the receiver to act simultaneously, so when the sender initiates the request, it needs to wait until the receiver responds. This means that the sender is blocked during the time when the receiver is proceeding with its request.
- **Asynchronous protocols:** Asynchronous protocols provide a non-blocking mode of communications. The sender initiates the request, but is not blocked during the waiting for the response and continues in its proceedings instead. This method is often referred to as the “*fire-and-forget model of communication.*” [56] The request can be performed later if the receiver decides to do so.

After reviewing the features of various communication protocols, it was decided to use the protocols which are described in further detail in this section.

2.6.1 HTTPS

During a discussion with the supervisor of this thesis, it was agreed that the entire application would be controlled from the browser via the web interface, which means that some variant of the HTTP should be used. Due to the confidentiality of the transmitted data and the privacy of the users [57], HTTPS is going to be used for communication between *Server Core* and the client device in the deployment.

2.6.2 Message queue

Since the application is divided into *the Server Core* and *the Scraper*, it is necessary to find a way in which *the Server Core* can send a scraping request to *the Scraper* module. Communication should be asynchronous because the scraping process can take a long time and *the Server Core* does not have to wait for it to complete because the public user who reports the website cannot see the results anyway.

The technology that meets these requirements is the message queue. This component can be described as a buffer that receives messages in a specific order and forwards them to the affected application while keeping the order unchanged. This allows for the decoupling of the sender and the recipient, who can work at their own pace and retrieve the messages from the queue only when they are ready to do so. [56]

RabbitMQ⁷ was chosen as the broker for message queue.

2.6.3 SFTP

Although there is already a database to store information about the reported websites, there is still a need to save larger files, such as embedded JavaScript source code or images from the website. Storing these types of file in the file system managed by the OS is usually more effective than storing them in the relational database, which is suited for smaller chunks of structured data. [58]

⁷<https://www.rabbitmq.com/>

One of the protocols that supports the transfer of files between two machines is SFTP⁸. It allows for operations such as file transfer, directory listings, or remote file removal, and the entire connection is encrypted thanks to the use of the SSH data stream. The client can authorize itself to the server using a password or SSH key, and the protocol is implemented on almost all major platforms. [59]

2.7 Programming language and framework

The already existing part of the project was written in Python using the Flask framework, so the main question was whether it should just be extended using existing technologies or rewritten in a different language from scratch.

After exploring the possible options, it was decided to rewrite the project in Kotlin with the Spring Boot framework. Rewriting an already established solution might seem like unnecessary work, but the size of the existing code base was relatively small, and Python was previously picked mainly because of its suitability for fast prototyping, so it was not insisted to keep the project's tech stack unchanged.

Kotlin is a modern, object-oriented programming language that supports interoperability with Java and therefore has support for a wide number of libraries and extensions. [60] Compared to Java, Kotlin provides null safety and a more concise syntax, although this is down to personal preference. The main reasons for choosing Kotlin over Python were static typing, better performance [61], and my greater familiarity with the JVM languages compared to the Python ecosystem.

Spring Boot is a pre-configured version of the Spring framework, which is suitable for the development of web-based applications and services. It comes with features such as the integrated web server, dependency injection, or externalized configuration. [62] This framework was chosen for its good documentation and the large number of extensions.

Gradle⁹ is used as a build system for the developed application.

⁸SSH File Transfer Protocol

⁹<https://kotlinlang.org/docs/gradle.html>

Implementation

This chapter takes the reader through the implementation of both application's components and describes the various problems faced during the process.

3.1 Data layer

The data layer is implemented in a separate Gradle module named *Common*, which is used by both *Server Core* and *Scraper* components. This solution makes it easy to edit application entities and database logic in one place, with the change propagated to the components that depend on it. The entities consist of `Author`, `DnsRecord`, `ModuleInfo`, `PhishingAccident`, `SslCertificate`, and `Website`.

Each entity is represented by a data class that is used in business operations throughout the entire application. The entity does not contain annotations or similar references to the database. Thanks to that, the code outside of the data layer is independent of the concrete implementation of the application's persistence logic.

```
data class DnsRecord(  
    override var id: Int?,  
    var name: String,  
    var type: Int,  
    var ipAddress: String,  
    var timeToLive: Long?,  
    var priority: Int? = 0,  
    var websiteId: Int = 0  
) : Model<Int>
```

■ **Listing 3.1** Example of the DNS record entity.

Every entity also has a corresponding object that represents the given entity in the database as a table. The columns of the table are defined by the DSL of the Exposed¹ framework, which takes care of mapping the object to the database table.

¹<https://github.com/JetBrains/Exposed>

```
object DnsRecords : IntIdTable() {
    val name = varchar("name", 30)
    val type = integer("type")
    val ipAddress = varchar("ip_address", 50)
    val timeToLive = long("ttl").nullable()
    val priority = integer("priority").nullable()
    val website = reference("website_id", Websites)
}
```

■ **Listing 3.2** Example of the table object for DNS records. The ID generation is implemented by the `IntIdTable` class from the Exposed framework.

The framework supports lightweight DAO² capability, which can generate SQL code for CRUD³ operations without the help from the developer. [63] The other option is to write these operations with Exposed DSL, which gives the developer more control about the actual SQL query that is going to be executed. The DSL approach was chosen for precisely this reason.

```
abstract class IntTableRepository<MODEL : Model<Int>, TABLE : IntIdTable> (
    val table: TABLE,
    val converter: RowConverter<MODEL>
) : CrudRepository<MODEL, Int> {
    override fun find(id: Int): MODEL? {
        val row = table.selectAll().where { table.id eq id }.singleOrNull()
        return if (row != null)
            converter.rowToRecord(row)
        else
            null
    }

    override fun findAll(): Collection<MODEL> {
        return table.selectAll().map { converter.rowToRecord(it) }
    }
}
```

■ **Listing 3.3** Part of the `IntTableRepository` abstract class which is used as a foundation for the repository operations. This class is then implemented for each entity and in this way new specific operations can be added.

3.2 Server Core

The Server Core component integrates Apache Tomcat⁴ web server, which handles incoming HTTP requests from the user or administrator.

It contains `RepositoryService` class with business operations. I would like to highlight the `getSimilarAccidents` method, which implements the required functionality of finding similar accidents from the past. It takes an instance of `PhishingAccident` as an argument, and returns the collection of accidents that is sorted by the number of modules which they share with the passed phishing accident. In this way, the administrator can view reported URLs that share a similar tech stack to the `PhishingAccident` that interests them.

²Data Access Object

³Create, Read, Update, Delete

⁴<https://tomcat.apache.org/>

The main page for the submission of phishing reports was taken from the existing part of the *phishingalert.cz* project and is marked as such in the project's source directory, while the control panel was developed as part of this thesis.

3.2.1 Control panel

The control (admin) panel is implemented as a part of Server Core component and uses Spring MVC⁵ framework. The View layer consists of Thymeleaf⁶ templates, which can be found in the `resources` directory in the Server Core project. These templates are filled with data from the Controllers, which are in the `cz.phishingalert.core.controllers.admin` package.

The pages of the control panel are styled with CSS from the Bootstrap v5.3⁷ library and the final design can be seen in the screenshots in figures 3.1 and 3.2

The panel can be accessed from the `/admin` URL path, from which the other actions can be performed through the GUI. The functionalities implemented in the control panel are the following:

- Viewing the list of phishing reports which are sorted by the submission ID, available from `/admin` path.
- Viewing the phishing reports which were submitted by the user with given email, available from `/admin/stats/user/<email>` path.
- Viewing the detailed statistics about the report with given ID and similar reports from the past if such reports exist, available from `/admin/stats/<ID>` path. The modules and DNS records shared between the viewed report and similar reports are marked with a green color on the page to visualize their equality.
- Sending the report to a higher authority, supported authorities being Google and Cloudflare. The report is send by accessing `/admin/submit/<authority>/<ID>` path.

3.2.2 Implementation of subsequent URL reporting

Another functionality is the reporting of obtained phishing URLs to the higher authorities. The application currently supports reporting to Cloudflare and Google Web Risk, which are analyzed in the Reporting tools subsection.

The Cloudflare reporting is controlled by the `CloudflareController` class and I have implemented it to demonstrate the possible way of reporting the URL to an authority without a public API for report submission. After the administrator clicks on the “Report website” button and chooses “Cloudflare” as an option, the server redirects them to the Cloudflare reporting form, which is already prefilled with the data from the original phishing author and with part of the data gained by the Scraper. This is done by embedding the information in the query parameters of the web form URL.


Reporting to Google Web Risk is implemented in the `WebRiskController` class, and presents the possibility of using the API for the submission of phishing reports. The report is send through the `WebRiskServiceClient`⁸ class of the Web Risk Java package, which utilizes the gRPC framework. With this framework, the developed application directly calls the methods on the remote machine in a similar way as if it were a local object. [64] Unfortunately, this method

⁵<https://docs.spring.io/spring-framework/reference/web/webmvc.html>

⁶<https://www.thymeleaf.org/>

⁷<https://getbootstrap.com/docs/5.3/getting-started/introduction/>

⁸<https://cloud.google.com/java/docs/reference/google-cloud-webrisk/latest/com.google.cloud.webrisk.v1>



Admin panel

ID	Website domain	Confirmed?	Date of report	Statistics	Report website
1	https://cvut.cz	<input type="checkbox"/>	17.04.2024 12:13:45	Statistics	Report website
2	https://cvut.cz	<input type="checkbox"/>	17.04.2024 14:59:23	Statistics	Report website
3	https://baedlung.com	<input checked="" type="checkbox"/>	17.04.2024 16:11:35	Statistics	Report website
4	https://fit.cvut.cz	<input type="checkbox"/>	19.04.2024 10:52:46	Statistics	Report website
5	https://fit.cvut.cz	<input type="checkbox"/>	19.04.2024 10:55:54	Statistics	Report website
6	https://fit.cvut.cz	<input type="checkbox"/>	19.04.2024 11:33:21	Statistics	Report website
7	https://fit.cvut.cz	<input type="checkbox"/>	19.04.2024 11:38:33	Statistics	Report website
8	https://wbvthdxsou.duckdns.org	<input type="checkbox"/>	19.04.2024 15:30:54	Statistics	Report website
9	https://seznam.cz	<input type="checkbox"/>	22.04.2024 18:35:25	Statistics	Report website
10	https://www.geeksforgeeks.org/	<input type="checkbox"/>	23.04.2024 23:01:29	Statistics	Report website
11	https://wikipedia.org	<input type="checkbox"/>	23.04.2024 23:03:48	Statistics	Report website
12	https://fit.cvut.cz	<input type="checkbox"/>	23.04.2024 23:05:07	Statistics	Report website
13	https://cvut.cz	<input type="checkbox"/>	23.04.2024 23:08:50	Statistics	Report website
14	https://cvut.cz	<input type="checkbox"/>	23.04.2024 23:10:47	Statistics	Report website
15	https://baedlung.com	<input checked="" type="checkbox"/>	23.04.2024 23:35:05	Statistics	Report website

■ **Figure 3.1** Screenshot of the control panel home page

Statistics for accident with ID: 64

<p>Author info</p> <p>Name: Jan Novák</p> <p>ID: 64</p> <p>Email: jan@gmail.com</p> <p>User agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:125.0) Gecko/20100101 Firefox/125.0</p> <p>IP: 127.0.0.1</p> <p style="text-align: center;">Reports from this author</p>	<p>Accident info</p> <p>URL: https://www.nachweisuberfalschgeld.com/</p> <p>Delivered at: 14.05.2024 16:17:22</p> <p>Phishing source: ff@dds.com</p> <p>Note: This is some example of note.</p> <p>Domain holder: "NACHWEISUBERFALSCHGELD.COM"</p> <p>Domain registered at: 146</p>
--	--

Similar reported accidents:

<p>Accident ID: 3</p> <p>Used modules:</p> <ul style="list-style-type: none"> • jQuery (Fast path), version: unknown • jQuery, version: 3.7.1 • Preact, version: 10 • WordPress, version: unknown <p>DNS records:</p> <ul style="list-style-type: none"> • baeldung.com., 172.66.40.248 • baeldung.com., 172.66.43.8 	<p>https://baeldung.com</p> <p style="text-align: right;">17.04.2024 16:11:35</p>
---	---

■ **Figure 3.2** Screenshot of the page with accident's statistics

could not be properly tested because the Web Risk submission API is not available for standard Google Cloud users, and even though I have made a formal request to Google to gain access to this feature, they did not manage to respond.

3.3 Scraper

The process of downloading the requested data from the entered URL is controlled from the `Orchestrator` class, which also coordinates the data exporting process.

The collections of entities, which are returned by the instances of `Downloaders` described in the related subsections, can be exported to the two sources: database and standard output stream. Exporting to the database is used when *the Scraper* is started in the standard way, and printing to standard output is used in the demo mode.

The website content, which consists of images, HTML, JavaScript, and CSS files, is stored in a temporary storage during the web crawling process, and then sent through SFTP via an instance of `SftpExporter` class to the server which is set in the configuration file of *the Scraper*. The SFTP connection is used only when the application is not running in the demo mode.

The demo mode is useful when it is necessary to test *the Scraper* without a running instance of *the Server Core* component. This mode is automatically entered if the administrator starts *the Scraper* with `--try-domain=<URL>` flag. In that case, communication with *the Server Core* through the message queue does not start, and the results of the scraping process for given URL are shown directly in a terminal and then discarded.

3.3.1 Obtaining domain information

As mentioned in the subsection Domain information, there are currently two protocols available that can be used to query the database with information about the registration of requested domain. It was decided to implement the database lookup using both of them, with RDAP being a primary protocol for this task and WHOIS being a fallback in case the RDAP lookup fails. Choosing this approach was a necessity caused by the fact that multiple domain registries still have not implemented the RDAP properly and instead rely on the old WHOIS protocol for serving the requested data. [65]

The domain information lookup is implemented in the `WebsiteDownloader` class, and the results are handled by the code from the `cz.phishingalert.scrapers.downloaders.parsers` package.

RDAP lookup is implemented using the `HttpClient` from the `java.net.http` package. The lookup function sends an HTTP GET request to the main RDAP server whose URL is set in the configuration file. If the request is successful, the server responds with a JSON file containing the information about the domain. This JSON file is then parsed into an instance of `Website` class and returned. The parsing process is skipped if the request to the RDAP server fails⁹, and the caller of the lookup function is informed about the failure.

The implementation of WHOIS lookup functionality is more complicated. Unlike RDAP, this protocol does not provide an option to automatically follow redirects to the server which stores the detailed information about the given domain. Instead, the server response includes the domain of the referred server in plain text, which can be seen in the example 3.4, and the client has to parse it. My WHOIS lookup algorithm uses the `WhoisClient` from the Apache Commons API¹⁰. The algorithm goes to the referred servers until it reaches the final destination or exceeds the upper limit of the visited servers. Then it attempts to parse the server response using a set of regular expressions and eventually saves it to an instance of `Website` class, which

⁹This usually happens when the domain registry has not implemented RDAP yet

¹⁰<https://commons.apache.org/proper/commons-net/apidocs/org/apache/commons/net/whois/WhoisClient.html>

is then returned. The parsing process is not 100% reliable due to the aforementioned lack of standardization of the WHOIS response format [11], but at least the responses from the major domain registries are properly parsed.

```
% IANA WHOIS server
% for more information on IANA, visit http://www.iana.org
% This query returned 1 object

refer:      whois.verisign-grs.com
...
```

■ **Listing 3.4** Example of part of the response from WHOIS lookup. The response from `whois.iana.org` contains a reference to another server that has more information about the queried domain.

3.3.2 Obtaining DNS records

The DNS lookup is implemented in the `DnsDownloader` class with the help of `dnsjava` library. Lookups are performed by querying the default DNS server that is set in the system settings, as this approach seemed more natural than defining this setting separately in the application's configuration.

The application currently examines DNS records of type *A*, *AAAA*, *MX*, *CNAME*, and *NS*, since these values are defined for most existing domains. [66] If needed, the range of downloaded DNS records can be extended by adding the missing types to the constructor of `DnsDownloader` and editing the `handleRecord` method.

3.3.3 Obtaining SSL certificates

The implementation of SSL certificate retrieval can be found in the `CertificateDownloader` class. The download process is performed through the Java SSL library¹¹ which obtains the entire certificate chain through a secure connection. The SHA-1 thumbprint is calculated for each certificate, and the parsed certificate data are subsequently stored in an instance of `SslCertificate` class.

3.3.4 Web crawler

The web crawler is implemented in the `cz.phishingalert.scrapers.crawler` package in the `PlaywrightCrawler` class, which extends the more generic `Crawler` abstract class that can be used as a basis for another web crawler implementation.

The whole crawling process is done according to the algorithm whose pseudocode can be seen in the listing 1. It is based on the BFS¹² algorithm with appropriate conditions for adding new URLs to the queue (line 17) and a small optimization which can save some time by not calling the browser API when not needed (line 12).

Each URL retrieved from the queue is handled by the `tryToNavigate` method, that tries to access the passed URL and changes the browser's User-Agent header in case of an unsuccessful attempt, which can be seen in the listing 3.5. The User-Agent switching is implemented in the `CrawlingProcess` helper class, which encapsulates the underlying browser instance.

¹¹<https://docs.oracle.com/javase/8/docs/api/javax/net/ssl/package-summary.html>

¹²Breadth-first search

Algorithm 1 Web crawling**Require:** $u \leftarrow$ starting URL

```

1:  $Q \leftarrow$  empty queue of URLs
2:  $S \leftarrow$  empty set of already found URLs
3: Insert  $u$  at the end of  $Q$ 
4: Insert  $u$  into  $S$ 
5:
6: while  $Q$  is not empty do
7:    $current \leftarrow$  first element from the  $Q$ 
8:   Remove first element from the  $Q$ 
9:
10:  Visit the page at  $current$  URL and download resources from it
11:  if number of elements in  $S \geq$  limit of visited pages then
12:    continue ▷ Optimization for skipping costly operation
13:  end if
14:
15:   $links \leftarrow$  URLs referenced from the  $current$  page
16:  for  $link$  in  $links$  do
17:    if # of elements in  $S <$  limit of visited pages AND  $S$  does not contain  $link$  then
18:      Insert  $link$  at the end of  $Q$ 
19:      Insert  $link$  into  $S$ 
20:    end if
21:  end for
22: end while

```

```

// Taken from the PlaywrightCrawler.kt
fun tryToNavigate(process: CrawlingProcess, url: URI) {
    if (process.page.url() == url.toString()) {
        logger.info("Crawler is already at the given $url")
        return
    }

    var triesCount = 1
    var pageResponse = process.page.navigate("$url")
        ?: throw PlaywrightException("...")

    while (triesCount <= config.triesPerPageLimit &&
        pageResponse.status() == HttpStatus.TOO_MANY_REQUESTS.value()) {
        process.switchUserAgent()
        pageResponse = process.page.navigate("$url")
            ?: throw PlaywrightException("...")
        triesCount++
    }

    // Handle the scenario where we ran out of tries and the website
    // still didn't let us in.
    if (pageResponse.status() == HttpStatus.TOO_MANY_REQUESTS.value())
        throw PlaywrightException("...")
}

```

■ **Listing 3.5** Example of the `tryToNavigate` method from the crawler which is used for accessing given URL.


```

crawler-config:
  browserProfilePath: /home/bob/firefox/common/.mozilla/firefox/profile
  visitedPagesLimit: 5 #Max number of visited websites
  triesPerPageLimit: 3 #Max number of attempts to navigate the same URL
  allowOutsideDomain: true #Visit links which lead to external websites
  userAgents:
    - "Mozilla/5.0 (X11; Linux x86_64; rv:124.0) Gecko/201001 Firefox/124.0"
    - "Mozilla/5.0 (Windows NT; Win64; x64; rv:124.0) Gecko Firefox/124.0"

```

■ **Listing 3.6** Example of the web scraper configuration in YAML format.

After that, the page’s HTML content, JavaScript files, CSS files, and images are downloaded to the local storage of the Scraper module. The crawler keeps finding new URLs on the page and then accessing them until the whole website is crawled or the upper limit of visited URLs is reached. The crawler also takes a screenshot of the page located at the starting URL.

The crawler settings can be adjusted by changing the values under `crawler-config` key in the Scraper module configuration file, as seen in the listing 3.6.

For example, the `browserProfilePath` setting provides an option to start the browser with an existing user profile, which is useful when there is a need to start the browser with some specific user setting or extension. I have observed that the crawling process is usually more reliable when the started browser uses an extension that automatically closes cookie pop-ups¹³. The screenshot of the visited URL looks clearer because the website content is not covered by the pop-up. The crawler also does not immediately start following the URLs referenced by the pop-up and follows the URLs referenced by the actual website itself, so the crawled results can be more relevant. Another useful option is the `userAgents` list, which can be extended by adding new strings in the YAML list format.

The upper limit of URLs visited in one crawling session can be set, and the same applies to the maximum number of attempts after which the browser stops trying to visit the given URL. The administrator can also allow or disable the crawling of domains different from the one of the starting URL.

3.3.5 Obtaining the information about used modules

The finding of information about the used modules is implemented in the `ModuleDownloader` class.


The process starts by visiting the reported URL with a browser controlled by the aforementioned Playwright library. After the page is ready, the algorithm loads the file with detection patterns written in JavaScript and evaluates this JavaScript code directly on the opened web page. The results are subsequently parsed and returned to the caller of the `download` method.

The definitions of detection patterns consist of a list of JavaScript functions that look for exposed global variables and functions that detect if the given technology is present. The file with these detection patterns was taken from an existing open-source project “*Library Detector For Chrome*¹⁴”, which is available under the MIT license. The `ModuleDownloader` is compatible with its return values, which are parsed in the `extractVersion` method. This means that the developed solution should work with future additions to the list of definitions from the “*Library Detector For Chrome*” project without any necessary adjustments.

The identified modules consist of various JavaScript libraries and frameworks, content management systems, and external APIs.

¹³<https://addons.mozilla.org/en-US/firefox/addon/istilldontcareaboutcookies/>

¹⁴<https://github.com/johnmichel/Library-Detector-for-Chrome>



Chapter 4

Testing

The application has been partially tested through unit and integration tests during the development process. The developed project was also reviewed with end-to-end tests that used real-world data to check the reliability and performance of the entire system. This chapter describes these tests in more detail.

4.1 Automated testing

Unit tests are used to test some of the helper methods and functions in *the Scraper* component, such as URL and date validation. They are also used to test the parsing of responses from the RDAP and WHOIS protocol. In case of WHOIS, they have proven to be a very useful tool, because the response format is not standardized, which was mentioned in the subsection 1.4.1. Due to this, the parsing process needed to be tuned and verified multiple times.

Integration tests are used only to a small extent to test classes that implement database operations, such as `AuthorRepository` or `WebsiteRepository`. The tests are performed by querying the H2¹ in-memory database, which implements all required SQL operations without the need to start a dedicated database server. These tests also check several business operations in the `RepositoryService` class.

4.2 End-to-end testing

The project has been tested mainly manually with end-to-end tests done on real websites from the Internet, which has seemed like a sensible solution because the application's use cases are heavily dependent on it.

However, even with the real data used, the performed tests can still give only a limited picture of the project's reliability. This is because the requests to the analyzed websites were made from residential IP from the network of a local Internet service provider, and therefore are probably less suspicious in the network traffic than the project's instance deployed in a large cloud, as was discussed in the IP address quality subsection.

¹<https://www.h2database.com/html/main.html>



■ **Figure 4.1** Graph (visible from Cloudflare dashboard after login) with the performed requests to the test page and their rating according to Cloudflare. Every rating which is not “Unsolved” is seen as a success.

4.2.1 Accessing websites with bot protection

To test how *the Scraper* performs against a website with bot protection enabled, I have created a simple test page and deployed it to Cloudflare Pages² hosting, which also provides a range of protections against automated visitors. One of them is the Cloudflare Turnstile, which is an alternative to the classic CAPTCHA. The user does not have to perform any demanding tasks as part of the anti-bot check, because their device is checked by the methods based on those listed in the Bot prevention section instead. [67] I have enabled it in the Cloudflare dashboard to see if it detects the application and marks it as a bot.

The test page is available at <https://learning-js.pages.dev/> and includes the jQuery³ library, CSS stylesheet, image, link to a different page under the same domain, and link to the external domain.

The test consisted of visiting the page multiple times with various types of the Cloudflare Turnstile turned on, and then manually checking if the application solved the anti-bot challenge and, therefore, passed as a real user. As can be seen in the figure 4.1, the application solved the challenge most of the time. The results of the test show that the developed solution has no problems with the easier non-interactive challenge, but is usually detected by the harder interactive challenge, so there is a room for improvement in *the Scraper's* crawling module.

The test can be potentially skewed by the other page visitors, but the number of application runs corresponded to the number of requests in the graph, so it should not be the case in this specific example.

4.2.2 Accessing potentially malicious websites

This task tested both *the Server Core* and *the Scraper* component. It consisted of creating a list of 20 potentially malicious websites from the PhishTank database and submitting them to *the*

²<https://pages.cloudflare.com/>

³<https://jquery.com/>

Server Core which handles the rest.

The website URLs have been manually selected from the <https://phishtank.org/> and subsequently stored in the enclosed text file, which is loaded by the simple Python script stored in the `report_sender.py`. This script then performs HTTP POST requests towards *the Server Core* and the results are manually checked after *the Scraper* finishes its job.

The results check consists of looking at the details about the downloaded websites in the database and in the control panel. Similar types of test were performed multiple times during the development process and it helped with the tuning of domain info parsing and website crawling.

Conclusion

The main goal of this thesis was the creation of the forensic module, which can obtain data related to the website and use it for comparison with previously reported websites. The next goal was the implementation of the administrator's control panel for further phishing reporting and website analysis.

The module was successfully implemented and provides the project administrator with the ability to get all the information about website registration, its DNS records, SSL certificates, and the frameworks and libraries used. It can also download all HTML, CSS, and JavaScript files used by the frontend of the reported website and transfer them via a secure protocol to the desired storage. The biggest obstacle was the need of not getting caught by the existing anti-bot protections, which are presented in the first chapter. With this in mind, an approach based on a headless browser was used and the whole solution was tested in multiple scenarios, which are described in the related chapter. The most serious drawback of this approach is the limited speed of the crawling process, although this has not been considered as a serious problem because of the low frequency of incoming reports. Should this become an issue, the project architecture provides the application with the ability to utilize multiple instances of such modules and therefore makes the whole service easily scalable in the future.

The administration panel was implemented in a form of web application which is rendered on a side of the server. It provides the administrator with the ability to easily report received websites to authorities such as Google or Cloudflare, and this list can also be extended in the future without bigger problems. The program can also show a detailed page about each reported website and find similar ones based on the modules used. The analysis of reported websites is currently limited and there is room for improvement which could utilize machine learning or advanced statistics to employ all the data gained by the module and predict the incoming phishing attacks.

When it comes to the previously existing part of the application, it was rewritten in Kotlin with Spring Boot framework which means that the whole project uses one established technology which can run on most of the modern systems without bigger adjustments.

Appendix A

Installation guide

The installation process was tested on Ubuntu 22.04 LTS, with Gradle version 8.5, Docker version 24.0.5., and OpenJDK 19.

The application was developed in IntelliJ IDEA 2023.3 (Ultimate Edition) and its run configuration is part of the project (stored in the `.run` subdirectory of both `core` and `scraper`). This means that if you have access to this IDE, you will be able to start and test the application relatively easily by opening both `core` and `scraper` in it and running it from there just with a click of a button without any manual Java/Gradle configuration needed. In that case you only need to run the `docker compose up` from the project's root directory and edit the SFTP server login detail in the `scraper/scraper-config.yml` YAML file before running the application.

In order to start the project and try both the Server Core and The Scraper locally, the following prerequisites are needed:

- Terminal with Bash (Unix Shell)
- Java 17 SDK or newer
- Gradle 8.5 or newer
- Docker 24.0 or newer
- Running SFTP server (its login credentials can be edited in the `scraper/scraper-config.yml` under the `sftp-config` key)
- Free localhost ports 5432 (PostgreSQL), 5672 (RabbitMQ) and 8080 (the Server Core component)

Now, to install and run the project without the use of IDE, you need to do the following steps:

1. Open the project files from the attached medium, or from the GitHub repository. In that case you can clone the repository with the following command:

```
git clone https://github.com/jirikx/phishingalert_cz_new.git
```
2. Start your SFTP server and add its login details into `scraper/scraper-config.yml`
3. When you are in the project root directory (`phishingalert_cz_new`), run the command: `docker compose up`, which will start the PostgreSQL database and RabbitMQ broker.
4. Now you need to open two subdirectories, `core` and `scraper`, each in its own terminal tab.

5. Set JAVAPATH environment variable to your Java SDK install location, for example

```
JAVAPATH=/home/bob/.jdk/openjdk-19
```

6. Then run this in the `core` directory to start the Server Core:

```
./gradlew -Dorg.gradle.java.home=$JAVAPATH bootRun --args='--spring.config.additional-location=file:./core-config.yml'
```

7. Now, run this in the `scraper` directory to start the Scraper:

```
./gradlew -Dorg.gradle.java.home=$JAVAPATH bootRun --args='--spring.config.additional-location=file:./scraper-config.yml'
```

8. Open `localhost:8080` to access the web form or `localhost:8080/admin` to access the control panel in your browser.

Bibliography

1. RUSHTON, Jo. *Phishing attacks statistics and facts 2024* [online]. 2024-03-01. [visited on 2024-03-28]. Available from: <https://www.techopedia.com/phishing-statistics>.
2. VERIZON. *What is Phishing? Definition, Types of Phishing, & Examples — Verizon* [online]. 2024. [visited on 2024-04-30]. Available from: <https://www.verizon.com/about/account-security/phishing>.
3. YEBOAH-BOATENG, Ezer Osei; AMANOR, Priscilla Mateko. Phishing, SMiShing & Vishing: An Assessment of Threats against Mobile Devices. *Journal of Emerging Trends in Computing and Information Sciences*. 2014, vol. 5, no. 4, pp. 299–300. Available also from: https://e-tarjome.com/storage/btn_uploaded/2020-09-12/1599891065_11216-etarjome%20English.pdf.
4. IBM. *What is a Phishing Attack — IBM* [online]. [N.d.]. [visited on 2024-04-30]. Available from: <https://www.ibm.com/topics/phishing>.
5. JAKOB_G. *YouTube doesn't want to take down scam ads [Reddit thread]*. *r/youtube* [online]. 2023-12-12. [visited on 2024-04-30]. Available from: https://old.reddit.com/r/youtube/comments/18gjiqy/youtube_doesnt_want_to_take_down_scam_ads/.
6. SECUREWORLD. *5 Smishing Attack Examples Everyone Should See* [online]. 2020. [visited on 2024-04-30]. Available from: <https://www.secureworld.io/industry-news/5-smishing-attack-examples-everyone-should-see>.
7. ALTEXSOFT. *Functional and Nonfunctional Requirements: Specification and Types* [online]. 2023. [visited on 2024-02-12]. Available from: <https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/>.
8. CLOUDFLARE, Inc. *What is a domain name? — Domain name vs. URL* [online]. 2024. [visited on 2024-05-10]. Available from: <https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name/>.
9. DAIGLE, Leslie. *WHOIS Protocol Specification* [RFC 3912]. RFC Editor, 2004 [visited on 2024-05-10]. Request for Comments, no. 3912. Available from DOI: 10.17487/RFC3912.
10. DOMAINTOOLS. *What is Whois Information and Why is it Valuable?* [online]. 2024. [visited on 2024-05-10]. Available from: <https://www.domaintools.com/support/what-is-whois-information-and-why-is-it-valuable/>.
11. INTERNET CORPORATION FOR ASSIGNED NAMES AND NUMBERS. *RDAP FAQs* [online]. 2018. [visited on 2024-05-10]. Available from: <https://www.icann.org/resources/pages/rdap-faqs-2018-08-31-en>.

12. CLOUDFLARE, Inc. *What is a domain name registrar?* [online]. 2024. [visited on 2024-05-10]. Available from: <https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name-registrar/>.
13. VAZQUEZ, Camilo Quiroz; GOODWIN, Michael. *What are DNS records?* [online]. IBM, 2024-01. [visited on 2024-05-10]. Available from: <https://www.ibm.com/topics/dns-records>.
14. MOCKAPETRIS, Paul. *Domain names - implementation and specification* [RFC 1035]. RFC Editor, 1987 [visited on 2024-05-10]. Request for Comments, no. 1035. Available from DOI: 10.17487/RFC1035.
15. WAYOFTHEPIE. *Structure of an SSL (X.509) certificate* [online]. DEV Community, 2020-05-22. [visited on 2024-05-14]. Available from: <https://dev.to/wayofthepie/structure-of-an-ssl-x-509-certificate-16b>.
16. CLOUDFLARE, Inc. *What is an SSL certificate?* [online]. 2024. [visited on 2024-04-25]. Available from: <https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/>.
17. CLOUDFLARE, Inc. *What is a reverse proxy? — Proxy servers explained* [online]. 2024. [visited on 2024-02-12]. Available from: <https://www.cloudflare.com/en-gb/learning/cdn/glossary/reverse-proxy/>.
18. STASSOPOULOU, A.; DIKAIAKOS, M.D. Crawler Detection: A Bayesian Approach. In: *International Conference on Internet Surveillance and Protection (ICISP'06)* [online]. 2006, pp. 16–16 [visited on 2024-05-04]. Available from DOI: 10.1109/ICISP.2006.7.
19. DIKAIAKOS, Marios D.; STASSOPOULOU, Athena; PAPAGEORGIU, Loizos. An investigation of web crawler behavior: characterization and metrics. *Computer Communications* [online]. 2005, vol. 28, no. 8, pp. 880–897 [visited on 2024-05-05]. ISSN 0140-3664. Available from DOI: <https://doi.org/10.1016/j.comcom.2005.01.003>.
20. CLOUDFLARE, Inc. *What is robots.txt? — How a robots.txt file works* [online]. 2024. [visited on 2024-05-05]. Available from: <https://www.cloudflare.com/learning/bots/what-is-robots-txt/>.
21. GOOGLE. *What is CAPTCHA?* [online]. 2024. [visited on 2024-02-12]. Available from: <https://support.google.com/a/answer/1217728>.
22. DZIEZA, Josh. *Why CAPTCHAs have gotten so difficult* [online]. The Verge, 2019-02 [visited on 2024-05-08]. Available from: <https://www.theverge.com/2019/2/1/18205610/google-captcha-ai-robot-human-difficult-artificial-intelligence>.
23. AMIN AZAD, Babak; STAROV, Oleksii; LAPERDRIX, Pierre; NIKIFORAKIS, Nick. Web Runner 2049: Evaluating Third-Party Anti-bot Services. In: MAURICE, Clémentine; BILGE, Leyla; STRINGHINI, Gianluca; NEVES, Nuno (eds.). *Detection of Intrusions and Malware, and Vulnerability Assessment* [online]. Cham: Springer International Publishing, 2020, pp. 135–159 [visited on 2024-05-04]. ISBN 978-3-030-52683-2. Available from: https://link.springer.com/chapter/10.1007/978-3-030-52683-2_7.
24. RICHABADAS, Tushar. *Threat Spotlight: How bad bot traffic is changing* [online]. Barracuda, 2023-10. [visited on 2024-05-08]. Available from: <https://blog.barracuda.com/2023/10/18/threat-spotlight-bad-bot-traffic-changing>.
25. BERN UNIVERSITY OF APPLIED SCIENCES. *abuse.ch — BFH* [online]. 2024. [visited on 2024-04-25]. Available from: <https://www.bfh.ch/en/research/reference-projects/abuse-ch/>.
26. URLHAUS. *URLhaus — About* [online]. 2024. [visited on 2024-04-25]. Available from: <https://urlhaus.abuse.ch/about/>.

27. URLHAUS. *URLhaus — API* [online]. 2024. [visited on 2024-04-25]. Available from: <https://urlhaus.abuse.ch/api/>.
28. URLHAUS. *URLhaus — Statistics* [online]. 2024. [visited on 2024-04-25]. Available from: <https://urlhaus.abuse.ch/statistics/>.
29. URLHAUS. *URLhaus — Browse* [online]. 2024. [visited on 2024-04-25]. Available from: <https://urlhaus.abuse.ch/browse/>.
30. PHISHTANK. *PhishTank ¿ Frequently Asked Questions (FAQ)* [online]. [N.d.]. [visited on 2024-04-28]. Available from: <https://phishtank.org/faq.php>.
31. PHISHTANK. *PhishTank ¿ API Information* [online]. [N.d.]. [visited on 2024-04-28]. Available from: https://phishtank.org/api_info.php.
32. PHISHTANK. *PhishTank ¿ Developer Information* [online]. [N.d.]. [visited on 2024-04-28]. Available from: https://phishtank.org/developer_info.php.
33. PHISHTANK. *PhishTank* [online]. [N.d.]. [visited on 2024-04-28]. Available from: <https://phishtank.org/>.
34. GOOGLE. *Google Safe Browsing — Google for Developers* [online]. [N.d.]. [visited on 2024-04-28]. Available from: <https://developers.google.com/safe-browsing>.
35. GOOGLE. *Report a Phishing Page* [online]. [N.d.]. [visited on 2024-02-13]. Available from: https://safebrowsing.google.com/safebrowsing/report_phish/?hl=en/.
36. GOOGLE. *Overview — Safe Browsing APIs (v4) — Google for Developers* [online]. 2021. [visited on 2024-04-28]. Available from: <https://developers.google.com/safe-browsing/v4>.
37. GOOGLE. *Pricing — Web Risk — Google Cloud* [online]. [N.d.]. [visited on 2024-04-28]. Available from: <https://cloud.google.com/web-risk/pricing>.
38. GOOGLE. *Safe Browsing — Google Safe Browsing* [online]. [N.d.]. [visited on 2024-02-13]. Available from: <https://safebrowsing.google.com/>.
39. CLOUDFLARE, Inc. *Abuse form — Cloudflare — The web performance & security company* [online]. 2024. [visited on 2024-04-25]. Available from: <https://abuse.cloudflare.com/phishing/>.
40. CLOUDFLARE, Inc. *Abuse approach - Cloudflare* [online]. 2024. [visited on 2024-02-13]. Available from: <https://www.cloudflare.com/en-gb/trust-hub/abuse-approach/>.
41. URLSCAN GMBH. *URL and website scanner* [online]. 2024-05-02. [visited on 2024-05-02]. Available from: <https://urlscan.io/>.
42. URLSCAN GMBH. *About* [online]. 2024-05-02. [visited on 2024-05-02]. Available from: <https://urlscan.io/about/>.
43. URLSCAN GMBH. *FAQ - Frequently Asked Questions* [online]. 2024-05-02. [visited on 2024-05-02]. Available from: <https://urlscan.io/docs/faq/>.
44. CLOUDFLARE, Inc. *URL Scanner* [online]. 2024. [visited on 2024-05-12]. Available from: <https://radar.cloudflare.com/scan>.
45. CARDITA, Sofia; MORARU, Alexandra. *Cloudflare's URL Scanner, new features, and the story of how we built it* [online]. Cloudflare, Inc., 2024-08. [visited on 2024-05-12]. Available from: <https://blog.cloudflare.com/building-urlscanner>.
46. WAPPALYZER. *Technology lookup* [online]. [N.d.]. [visited on 2024-05-12]. Available from: <https://www.wappalyzer.com/lookup/>.
47. WAPPALYZER. *Find out what CMS or framework a website is using* [online]. [N.d.]. [visited on 2024-05-12]. Available from: <https://www.wappalyzer.com/articles/find-out-what-cms-or-framework-a-website-is-using/>.

48. WAPPALYZER. *Pricing* [online]. [N.d.]. [visited on 2024-05-12]. Available from: <https://www.wappalyzer.com/pricing/>.
49. SHELDON, Robert. *What is model-view-controller (MVC)?* [online]. TechTarget, 2023-09. [visited on 2024-05-06]. Available from: <https://www.techtarget.com/whatis/definition/model-view-controller-MVC>.
50. GOOGLE. *Google Terms of Service - What we expect from you* [online]. 2022. [visited on 2024-05-05]. Available from: <https://policies.google.com/terms?hl=en#toc-what-we-expect>.
51. DATA TOGETHER. *Comparison of web archiving software* [online]. GitHub, 2017-08-25. [visited on 2024-05-11]. Available from: https://github.com/datatogether/research/tree/master/web_archiving.
52. MICROSOFT. *Playwright enables reliable end-to-end testing for modern web apps* [online]. 2024. [visited on 2024-05-11]. Available from: <https://playwright.dev/java/>.
53. AMAZON WEB SERVICES, INC. *What's the Difference Between Relational and Non-relational Databases?* [online]. 2024. [visited on 2024-05-06]. Available from: <https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/>.
54. SCYLLADB. *What is Eventual Consistency? — Definition and F&Qs* [online]. 2024. [visited on 2024-05-06]. Available from: <https://www.scylladb.com/glossary/eventual-consistency/>.
55. KUL, Atakan. *Microservice Communication (Synchronous vs Asynchronous)* [online]. Medium, 2023-06-15. [visited on 2024-05-08]. Available from: <https://atakankul.medium.com/microservice-communication-synchronous-vs-asynchronous-91b31670b3c6>.
56. BANERJEE, Deboshree; AIBIN, Michal. *Introduction to Message Queues* [online]. Baeldung, 2024. [visited on 2024-05-08]. Available from: <https://www.baeldung.com/cs/message-queues>.
57. H, Jamie. *Serve websites over HTTPS (always): You should be serving web pages over HTTPS. Are you?* [online]. National Cyber Security Centre, 2018-06-06. [visited on 2024-05-16]. Available from: <https://www.ncsc.gov.uk/blog-post/serve-websites-over-https-always/>.
58. GRAY, Jim; INGEN, Catharine van; SEARS, Russell. *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem* [online]. Microsoft Research and University of California at Berkeley, 2006-04. [visited on 2024-05-01]. Tech. rep., MSR-TR-2006-45. Available from: <https://www.microsoft.com/en-us/research/publication/to-blob-or-not-to-blob-large-object-storage-in-a-database-or-a-filesystem/>.
59. SMALLCOMBE, Mark. *The What's, How's and Why's of SFTP* [online]. Integrate.io, 2023-09-21 [visited on 2024-05-08]. Available from: <https://www.integrate.io/blog/the-whats-hows-and-whys-of-sftp/>.
60. AKHIN, Marat; BELYAEV, Mikhail, et al. *Kotlin language specification: Kotlin/Core* [online]. JetBrains / JetBrains Research, 2020 [visited on 2024-05-07]. Available from: <https://kotlinlang.org/spec/introduction.html>.
61. ATTRACTIVECHAOS. *Programming Language Benchmark v2 (plb2)* [online]. GitHub, 2024-01-23 [visited on 2024-05-07]. Available from: <https://github.com/attractivechaos/plb2/blob/master/README.md>.
62. INC., Broadcom. *Spring — Why Spring?* [online]. 2024. [visited on 2024-05-08]. Available from: <https://spring.io/why-spring>.

63. STALLA, Alessio; AIBIN, Michal. *Guide to the Kotlin Exposed Framework* [online]. Baeldung, 2022. [visited on 2024-05-07]. Available from: <https://www.baeldung.com/kotlin/exposed-persistence>.
64. GOOGLE. *gRPC overview* [online]. 2024-05-09. [visited on 2024-05-14]. Available from: <https://cloud.google.com/api-gateway/docs/grpc-overview>.
65. BROWN, Gavin. *RDAP deployment dashboard* [online]. RDAP.org, 2024-05-11. [visited on 2024-05-11]. Available from: <https://deployment.rdap.org/>.
66. MLYTICS. *What are the most common types of DNS records?* [online]. 2024. [visited on 2024-05-13]. Available from: <https://learning.mlytics.com/domain-name-system/common-types-of-dns-records/>.
67. CLOUDFLARE, Inc. *Cloudflare Turnstile* [online]. 2024-04-22. [visited on 2024-05-14]. Available from: <https://developers.cloudflare.com/turnstile/>.

Attachment Contents

readme.txt.....	the file with attachment contents description
phishingalert_cz_new.....	implementation source code
├── common.....	common source code for both components
├── core.....	Server Core source code
├── scraper.....	Scraper source code
├── docker-compose.yml.....	compose file for DB and MQ
├── LICENSE.....	code license
├── README.md.....	short installation guide
└── tests.....	resources used for the testing of application
thesis.....	the thesis text directory
├── konvij11-thesis.pdf.....	the thesis text in PDF
└── src_thesis.....	L ^A T _E X source of the thesis text