INTERNATIONAL HELLENIC UNIVERSITY
DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

DIPLOMA THESIS

Automated Identification and Reporting of Suspicious/Phishing Websites: A Public and Free-to-use Service

Students:
Alexandros Magos
Student ID: 185320
Dimitrios Koutsoupias
Student ID: 185204

Supervisor:
Antonis Sidiropoulos

23 May 2023

Title of Dissertation Automated Identification and Reporting of Suspicious/Phishing Websites: A Public and Free-to-use Service
Code of Dissertation 23101
Student's full name Magos Alexandros
Student's full name Koutsoupuias Dimitrios
Supervisor's full name Sidiropoulos Antonis
Date of undertaking 14-01-2023
Date of completion 23-05-2023

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Dedication

We, the authors, wish to dedicate this paper to our families, who have been our pillar of strength, resilience, and motivation. Their ceaseless encouragement, backing, and understanding throughout this multifaceted journey have been instrumental in bringing our vision to fruition. Their unrelenting support during the long hours of exploration and composition, along with their generous investment in our wellbeing, has been an invaluable part of this process.

Subsequent to this, we extend a heartfelt dedication to our supervising professor, Dr. Antonis Sidiropoulos. His magnanimous commitment to this venture, complemented by unwavering mentorship, support, and priceless insights, has been pivotal in actualizing this intricate work. Always ready with an affable smile, he has guided us through this labyrinthine investigation and proffered aid along the sinuous path.

Our deepest gratitude goes out to everyone who has been a part of this endeavor. Your belief in our aspiration to contribute to the battle against malicious websites, striving for a safer internet for everyone, has made this all possible. Thank you.

Prolog

In the relentless battle against malicious websites, we must acknowledge the vital role that diverse strategies and techniques play. The pervasive presence of nefarious websites, notorious for partaking in deceitful actions, disseminating malicious software, and exfiltrating sensitive personal data, constitutes a significant peril for the global community of internet users. The continuous advancement of web technologies and the increasing complexity of cyber threats have led to the recognition of detecting and mitigating risks associated with websites as a crucial research domain.

The present study introduces an artificial intelligence (AI)-driven system designed to identify malicious websites. The system employs modern machine learning methodologies to provide a dynamic and continuously enhancing solution. Our system pinpoints potentially harmful websites by scrutinizing various features such as URL structure, content, and additional metadata. Not only do we shed light on our system's technical aspects, but we also delve into the merits of AI-driven models for this task, juxtaposing our approach with other prominent systems in the field.

By providing an accessible API and laying the groundwork for open-sourcing our project, we aspire to stimulate collaboration and spur the development of supplementary tools and methodologies that can bolster the realm of malicious website detection. As the internet persistently grows and transforms, our defenses against the omnipresent cyber threats must keep pace. This paper embodies both a contribution to this ongoing crusade and a rallying cry for researchers and practitioners to band together in pursuit of a more secure and safer online environment.

Summary

In this thesis, we unveil an AI-based system for detecting malicious websites, harnessing state-of-the-art machine learning techniques to deliver an adaptive and ever-improving solution. Our system pinpoints potentially harmful websites by scrutinizing various features such as URL structure, content, and additional metadata. Not only do we shed light on our system's technical aspects, but we also delve into the merits of AI-driven models for this task, juxtaposing our approach with other prominent systems in the field.

By providing an accessible API and laying the groundwork for open-sourcing our project, we aspire to stimulate collaboration and spur the development of supplementary tools and methodologies that can bolster the realm of malicious website detection. As the internet persistently grows and transforms, our defenses against the omnipresent cyber threats must keep pace.

Abstract

EN

This research paper presents the development of an AI-driven system for detecting malicious websites using machine learning techniques to analyze various website features. The aim of this study is to enchance internet security by delivering a user-friendly and easily accessible instrument that swiftly detects potential hazards. The system comprises a responsive frontend web application and a backend API, with a proposed future collaboration with law enforcement and CERT teams. Users can also report and vote on malicious websites, ensuring the system stays current with emerging threats.
The paper details the methodology, data acquisition, and analysis approaches employed in devising the system, as well as the appraisal of distinct machine learning models based on accuracy, precision, recall, F1-score, prediction time, and model dimensions. The chosen model exhibits its effectiveness when contrasted with established methods and instruments for detecting malicious websites. The outcomes emphasize the potential worth of the proposed system in enhancing internet safety, aiding readers in comprehending its primary features and implications for both researchers and users.

EL

Αυτή η ερευνητική εργασία παρουσιάζει την ανάπτυξη ενός συστήματος που ελέγχεται από την τεχνητή νοημοσύνη για τον εντοπισμό κακόβουλων ιστότοπων χρησιμοποιώντας τεχνικές μηχανικής μάθησης για την ανάλυση διαφόρων χαρακτηριστικών ιστότοπων. Στόχος αυτής της μελέτης είναι να ενισχύσει την ασφάλεια του διαδικτύου παρέχοντας ένα ευχρηστο και εύκολα προσβάσιμο εργαλείο που ανιχνεύει γρήγορα πιθανούς κινδύνους. Το σύστημα περιλαμβάνει μια ανταποκριτική εφαρμογή web και μια backend API, με μια προτεινόμενη μελλοντική συνεργασία με ομάδες επιβολής του νόμου και CERT. Οι χρήστες μπορούν επίσης να αναφέρουν και να ψηφίζουν για κακόβουλους ιστότοπους, εξασφαλίζοντας ότι το σύστημα παραμένει ενημερωμένο με τις εμφανιζόμενες απειλές.
Η εργασία αναλύει τη μεθοδολογία, την απόκτηση δεδομένων και τις προσεγγίσεις ανάλυσης που χρησιμοποιήθηκαν στη διαμόρφωση του συστήματος, καθώς και την αξιολόγηση διάφορων μοντέλων μηχανικής μάθησης με βάση την ακρίβεια, την ανάκληση, το σκορ F1, τον χρόνο πρόβλεψης και τις διαστάσεις του μοντέλου. Το επιλεγμένο μοντέλο επιδεικνύει την αποτελεσματικότητά του όταν αντιπαρατίθεται με εδραιωμένες μεθόδους και εργαλεία για τον εντοπισμό κακόβουλων ιστοτόπων. Τα αποτελέσματα τονίζουν την πιθανή αξία του προτεινόμενου συστήματος στην ενίσχυση της ασφάλειας στο διαδικτύου, βοηθώντας τους αναγνώστες να κατανοήσουν τα κύρια χαρακτηριστικά του και τις επιπτώσεις του για τους ερευνητές και τους χρήστες.

Contents

Chapter 1

Introduction

1.1 What is Cybercrime

A cyber attack is defined as a combination of tools and techniques utilized by an attacker to successfully execute an assault on their chosen target [1]. In the context of the online world, the internet is the primary weapon. The targets can be specific individuals or their digital devices.

The motives behind these attacks are diverse, encompassing economic, religious, and political reasons [1]. Espionage also plays a significant role, occurring at both national and individual levels. It's important to note that not all attacks immediately impact end users. Subtle, data-stealing assaults exploit vulnerabilities in web and database systems to extract user information. Subsequently, attackers may sell this information, fabricate false identities, or engage in blackmail for financial gain [2].

The internet, while highly convenient for communication, also provides a veil of anonymity that empowers these attackers. Individuals who possess skills in the physical world are more likely to conduct online attacks, and vice versa.

As advancements in artificial intelligence (AI) technology continue, cyber threats and scams evolve in tandem. AI-driven models can mimic user behaviors, generate fraudulent content and websites, and automate complex, multi-faceted attacks. A common example of this is AI-generated emails that appear to be legitimate. These AI models can exploit leaked emails, identify individuals on social media, and develop intricate profiles for the purpose of deception and blackmail.

It is therefore advised to exercise caution when sharing information on social media. AI technology can facilitate cybercrimes such as voice mimicry for the purpose of deception. For instance, in 2019, the CEO of a UK energy firm was tricked into transferring 220,000 euros following a conversation with an AI-generated voice that he believed was his superior [3]. In another case, a perpetrator bypassed bank login protection that utilized voice recognition technology with the help of freely available tools [4].

As technology progresses, vigilance against emerging threats, including early indicators of future attacks, is essential. By staying informed and cautious, we can better safeguard ourselves and our online assets from cybercriminals.

1.2 The Significance of Detecting and Preventing Malicious Websites

As we plunge headlong into the ever-morphing digital landscape, nefarious websites have proliferated at an alarming rate, ensnaring internet users in their virtual snares. With a staggering 4.6 billion souls connected to the World Wide Web, one cannot overlook the burgeoning rise of malicious sites preying on unsuspecting victims. According to the FBI's Internet Crime Complaint Center (IC3), a jaw-dropping $10.3 billion[5] vanished into the abyss of cybercrimes in the United States during the year 2021. Regrettably, this staggering sum ascends relentlessly with each passing annum, and the top crime for the last five years, phishing, has also rapidly increased year by year.

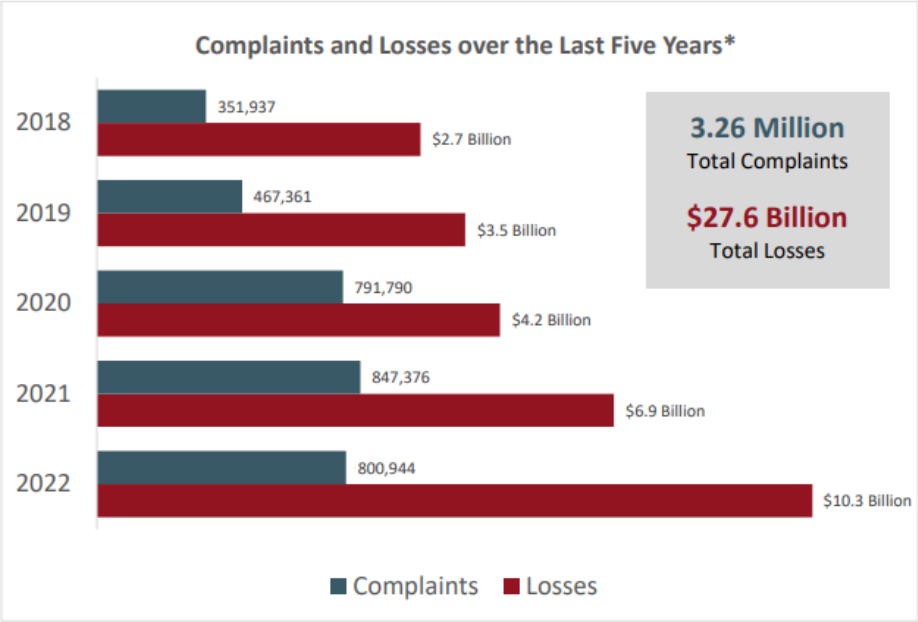

Figure 1.1: FBI's 2022 Internet Crime Complaint Center (IC3) Annual Internet Crime Report [5]

In the quagmire of the internet, one might stumble upon pernicious phishing sites - devious digital traps designed to filch sensitive personal data. Another scourge, the insidious drive-by download site, stealthily deposits malicious software onto users' devices, oftentimes unbeknownst to the victim. Google, the omnipresent search engine behemoth, disclosed in 2020 that it detected a mind-boggling 40,000 fresh phishing sites weekly. To compound matters, crippling ransomware attacks - such as the notorious WannaCry and NotPetya incidents - have left indelible scars on individuals, businesses, and even vital infrastructure.

In light of the inestimable damage that malevolent websites can inflict, it is imperative that we hone

our collective abilities to identify and thwart these virtual threats, thus shielding internet denizens and their precious information from harm.

1.2.1 Analyzing the Top Internet Crimes

In this subsection, we delve into the most prevalent internet crimes, including phishing, personal data breach, non-payment/non-delivery, extortion, and tech support scams, among others. Each of these crimes possesses unique characteristics and presents specific challenges for detection and prevention.

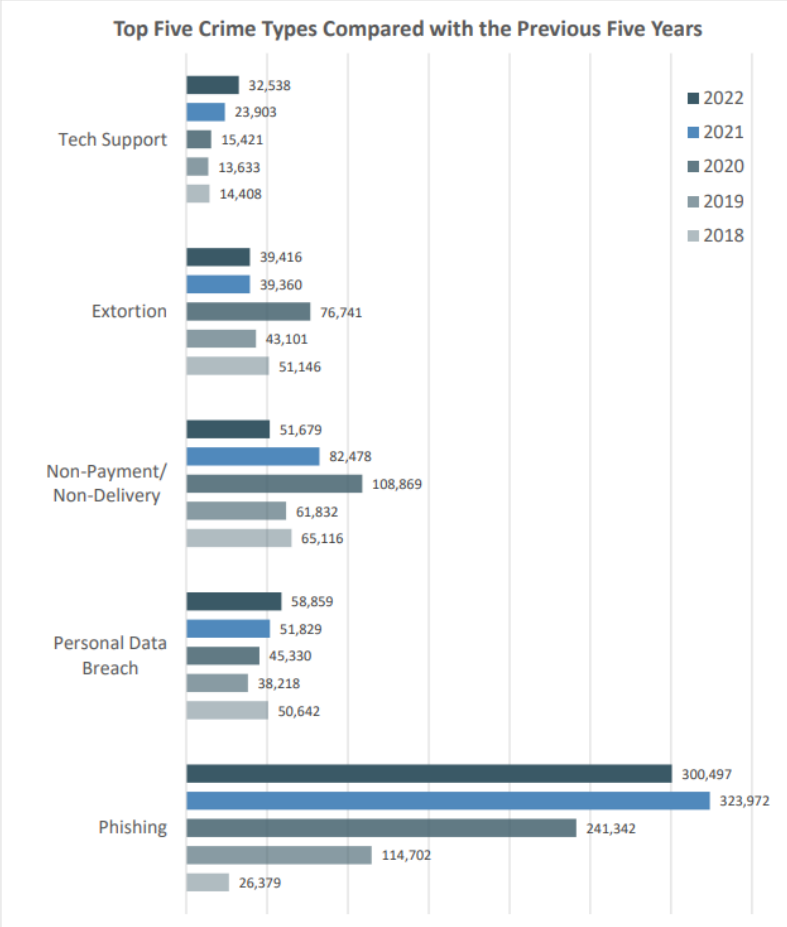

Figure 1.2: FBI's 2022 Internet Crime Complaint Center (IC3) Top Five Crimes [5]

The terms shown in the figure 1.2:

- **Phishing:** Phishing, a notorious online scam, involves cybercriminals masquerading as genuine entities to deceive users into divulging sensitive data, like login credentials or financial details. Often employing email or malicious websites that closely mimic legitimate counterparts, phishing attacks have surged in recent years, posing a significant danger to individuals and organizations alike.

- **Personal Data Breach:** Personal data breaches entail unauthorized access or exposure of an individual's private information held by an organization. Such breaches can stem from weak security measures, human error, or targeted cyberattacks. The repercussions can be severe, including identity theft, financial fraud, and reputational damage for the organization.
- **Non-payment/Non-delivery:** Non-payment and non-delivery scams encompass transactions in which one party neglects to deliver promised goods or services, or the other party fails to fulfill their payment obligations. These scams can manifest in various forms, such as online auction fraud, counterfeit merchandise sales, or fraudulent payment for services. They remain a major source of financial loss for internet users.
- **Online Extortion:** Online extortion involves obtaining money, property, or services from an individual or organization through coercion or threats. In the realm of internet crimes, extortion often entails ransomware attacks, where culprits encrypt a victim's data and demand payment for the decryption key. Other forms of cyber extortion include threats to disclose sensitive information or initiate a cyberattack unless payment is made.
- **Tech Support:** Tech support scams involve fraudsters posing as technical support representatives to trick victims into providing access to their devices or personal data. These cons often feature unsolicited phone calls or pop-up messages, warning users about non-existent issues with their computers or software. Once the victim is convinced, scammers may request remote device access, payment for unnecessary services, or installation of malicious software.

In addition to the attacks listed above, there are others not depicted in figure 1.2. Some of these attacks may be carried out via a malicious website, while others may be carried out via malware download. In more detail we have:

- **Malware attack:** In this situation, the user installs a malicious application on the victim's computer that impersonates a malicious user. Downloading dangerous programs is primarily accomplished through the use of applications that are, as they say, pirated scripts that may contain malicious software. This type of material can be obtained through malicious websites that deal with software piracy. Because it is practically software theft, software piracy is also a cyber security risk.
- **Insider attacks :** These are people who may or may not disclose certain confidential information that, if misused, has the potential to harm another party's reputation. However, it is not simply information sharing from mouth to mouth, but it can also be an overheard or voluntary act that creates a security vulnerability in a company's systems for the goal of some data leak.
- **Zero day attacks :** A zero-day attack is one that takes advantage of insecure software. More specifically, vulnerabilities that have not yet been patched, making them extremely dangerous. The day you discover this security flaw in the system, the manufacturer realizes the flaw exists in their system. In this way, we can see how difficult it is to be totally insured against this type of attack. The tremendously hazardous aspect of these types of attacks is

that, due to the nature of the attack, they can happen to nearly any organization, including very well-known ones. We can't determine for sure what kind of attack the application can launch on a service because they all differ and vary.

- **XSS attacks :**An xss attack is a type of attack in which the attacker uses the way the code of a page is written to run malicious code on a page that is not his and opens the page. In further depth, these are sites that can and have ways for visitors to input text to the page, such as websites that accept comments. A malicious user can install harmful code inside a website in this manner, and the code will run from the page, which may or may not have anything to do with the attacker.

A shared aspect among the internet crimes mentioned above, and numerous others, is the dependence on malicious websites crafted by threat actors to enable their malicious activities. These websites often function as communication channels, platforms for launching attacks, or storage for stolen data. For instance:

- In phishing attacks, malicious websites frequently impersonate genuine sites, tricking victims into disclosing sensitive data.
- Personal data breaches may arise from hackers exploiting website vulnerabilities to gain unauthorized access to user information.
- Non-payment/non-delivery scams could involve creating fake online stores or auction sites to deceive victims.
- Ransomware attacks and extortion typically need a malicious website for delivering instructions or processing ransom payments.
- Tech support scams depend on misleading websites to show false warning messages or enable remote access to victims' devices.

Considering the critical role malicious websites play in facilitating various internet crimes, effective detection and prevention of these sites become essential. By creating robust systems capable of accurately identifying and neutralizing malicious websites, we can significantly hinder cybercriminals' efforts and shield internet users from falling prey to these offenses. This highlights the importance and urgency of our research into developing a machine learning-based system for detecting and preventing malicious websites.

1.3 Defining the Problem: Classifying Websites Using Machine Learning

The specific problem addressed in this study is the development of a machine learning-based system to classify websites as malicious or benign. In recent years, machine learning has emerged as a

powerful tool for tackling complex problems, including the detection of malicious websites. This project aims to leverage machine learning techniques to improve detection accuracy, minimize false positives, and reduce computational complexity.

Our approach involves using a variety of website features, such as domain information, server type, and security certificates, to build a model that can accurately classify websites. By harnessing the power of machine learning algorithms, we aim to develop a system that outperforms traditional rule-based approaches in detecting malicious websites. This will help protect users from cyber threats and enable organizations to implement better security measures.

1.4 Objectives of the Study

The primary objectives of this study are:

1. To investigate the effectiveness of various machine learning algorithms in classifying websites as malicious or benign, based on their features.
2. To optimize the chosen model to minimize false positives and false negatives, ensuring a high level of detection accuracy.
3. To evaluate the computational complexity of the proposed solution and identify opportunities for reducing the processing time required to classify websites.

These objectives will guide the research and inform the selection of appropriate methods and models to develop an efficient and accurate malicious website detection system.

Chapter 2

Literature Review

The section aims to conduct a comparative analysis of the current solutions available in the market. The analysis will facilitate the identification of areas that require improvement and innovation for our system. Additionally, it will emphasize the significance of a varied range of tools to tackle the ever-changing landscape of cyber threats. With this perspective in mind, let's delve into the comparisons between our system and three well-known systems in the field: VirusTotal URL Checking, ImmuniWeb, and Urlscan.io.

2.1 Comparison with Other Methods

In this part, we compare our bad website detection system with three well-known systems found in research: VirusTotal URL Checking, ImmuniWeb, and Urlscan.io. By looking at their features and limits, we can better understand the special benefits of our approach.

2.1.1 VirusTotal URL Analysis

VirusTotal URL Checking is a popular service for finding harmful URLs. Their system mainly focuses on gathering data from many antivirus engines and website scanners, making it a shared approach instead of using AI-based detection models like our service. This is one of the most significant way in which the two systems are different..

When talking about their API, VirusTotal offers both free and premium versions. The free version is restricted to 500 daily queries at a rate of 4 per minute. It also can't be used in business products or services or in work processes that don't add new files. On the other hand, our system gives a more open API without these limits, supporting more use and teamwork.

2.1.2 ImmuniWeb

ImmuniWeb is another well-known tool for finding bad websites. While it has a similar goal as our service, the way it does things is very different. ImmuniWeb uses a scoring system based on different factors and limits, rather than AI-based detection models like ours. This different way may give different results, but it also shows the many methods available for solving the problem of bad website detection.

When it comes to their API, ImmuniWeb gives limited access for free users. Users may take 10 tests per day without an account, and 20 tests per day with an account. They also offer paid plans, with their first package allowing 50 tests per day at a cost of $200 per month. In contrast, our service gives a more open API without such strong limits, making it more interesting to a larger group of people.

2.1.3 Urlscan.io

Urlscan.io is another famous tool for finding bad websites. Like ImmuniWeb, it collects interesting factors and uses custom limit values to give a score. However, it does not use AI for its detection process, making its way of doing things very different from ours.

As for their API, Urlscan.io gives limited access that is usually more than enough for personal users. For non-personal or commercial use, a monthly license is required wich starts at $500. Our service, compared to this, gives a more open API without these limits, making it a better choice for a wider range of users.

In the end, while VirusTotal URL Checking, ImmuniWeb, and Urlscan.io are helpful tools in the fight against bad websites, our AI-based detection system offers some unique benefits. Our service focuses on machine learning models to give a more changing and always getting better solution. Also, our API is more open and less limiting, helping more teamwork and use among the community. This mix of features makes our system different, making it a strong and new way to find bad websites.

Chapter 3

Methodology

Prior to plunging into the intricate particulars of our methodology, let's first demystify the essential jargon, programming languages, libraries, and notions that pervade this segment. By doing so, we'll empower readers—technical maestros and laypeople alike—to grasp the underpinnings of the system architecture and its artful design more effectively.

3.1 Programming Languages

1. **Python[6]:** The Python programming language, admired for its versatility and remarkable simplicity, has etched a name for itself in the hearts of developers. A high-level language, it shines in realms such as web development, data analysis, and the ever-evolving machine learning landscape. Let's delve into the myriad aspects that render Python a perfect fit for this project.

    - **Python's elegant, streamlined syntax and emphasis on whitespace foster a pleasant reading experience, expediting the coding process.** The language boasts a vast standard library, teeming with pre-built modules and functions that obviate the need for laborious, ground-up coding.
    - **Not to be overlooked, Python's zealous community never ceases to contribute to the language's growth, proffering a treasure trove of resources, tutorials, and unwavering support.** Compatibility is Python's forte, as it melds seamlessly with diverse operating systems—Windows, macOS, and Linux—facilitating hassle-free application development and deployment across platforms.
    - **But wait, there's more!** Python's rich ecosystem is home to a dazzling array of libraries and frameworks, each meticulously crafted for specific tasks. Data manipulation enthusiasts can turn to NumPy and pandas, while machine learning aficionados are spoilt for choice with TensorFlow and PyTorch. Web development? Django and Flask have got you covered.

In the project at hand, Python takes center stage for both the training script and machine learning model development, capitalizing on its formidable libraries.

2. **JavaScript (Node.js)[7]:** JavaScript, a programming language that has long dominated the web development sphere, initially thrived as the go-to choice for client-side scripting in web browsers. Its evolution, however, ushered in Node.js—a game-changing runtime environment that opened the doors to server-side programming with JavaScript. Let's explore what makes this dynamic duo an ideal fit for the project at hand.

    - **With JavaScript, developers can wield a single language for both frontend and backend development, streamlining the process and obviating the need to master multiple languages.** JavaScript and Node.js inherently embrace asynchronicity and event-driven paradigms, paving the way for efficient management of numerous concurrent connections and bolstering application performance.
    - **Node.js was engineered with scalability in mind, making it an excellent choice for crafting high-performance web services.** Furthermore, its expansive ecosystem brims with libraries and frameworks that cater to specific needs, such as Express.js for web application development and Socket.IO for real-time communication.
    - **The JavaScript and Node.js community is nothing short of dynamic and engaged, tirelessly contributing to ongoing development and proffering invaluable resources, tutorials, and support.** For this project, Node.js takes the reins in constructing the backend API, capitalizing on its innate efficiency and scalability to build a high-performance server adept at handling a multitude of simultaneous connections.

3.1.1 Libraries and Frameworks

- **Pandas:** This potent Python library excels in data manipulation and analysis. With its signature DataFrame data structure, working with structured data becomes a breeze. Data cleaning, aggregation, transformation, and visualization are now within easy reach for data scientists and analysts. In our project, Pandas takes charge of reading and prepping CSV data, merging datasets, and cherry-picking model features.

- **Scikit-learn:** A comprehensive Python library for machine learning and data mining, Scikit-learn is equipped with an arsenal of tools for data preprocessing, model selection, training, and evaluation. Its consistent API and thorough documentation make it a developer and researcher favorite. In our project, Scikit-learn tackles data preprocessing, dataset splitting, Random Forest classifier training, and performance evaluation.

- **OneHotEncoder & SimpleImputer:** Hailing from the Scikit-learn library, these data preprocessing techniques pack a punch. OneHotEncoder converts categorical features into binary vectors, while SimpleImputer fills numerical gaps with a designated constant or strategy. Both techniques are employed during our project's data preprocessing phase.

- **Random Forest:** This machine learning algorithm harnesses the power of multiple decision trees, blending their predictions for accurate, robust outcomes. As an ensemble learning

method, it taps into the wisdom of the crowd, curbing overfitting and enhancing generalization. In our project, Random Forest trains the malicious website-detecting classifier.

- **Node.js:** Already introduced, Node.js is a server-side JavaScript runtime environment that facilitates the development of scalable, high-performance web applications. In our project, Node.js constructs the backend API.
- **Bootstrap 5 [8]:** This popular CSS framework streamlines the design of responsive, mobile-first web applications. Boasting a suite of pre-built components, Bootstrap 5 expedites web development and guarantees consistency across devices and screen sizes. In our project, it crafts a sleek, responsive frontend web application.
- **jQuery[9]:** As a nimble, lightweight JavaScript library, jQuery simplifies HTML document traversal, manipulation, and event handling. Its clean syntax eases HTML document work, paving the way for dynamic, interactive web applications. In our project, jQuery oversees AJAX requests and user interface updates sans page reloads.
- **EJS[10]:** Embedded JavaScript (EJS) is a straightforward templating language that generates HTML markup via JavaScript. By embedding JavaScript in HTML templates, EJS enables dynamic content rendering—an ideal solution for data-driven web applications. In our project, EJS masterfully creates frontend web application templates, seamlessly integrating dynamic content and UI components.

3.1.2 Concepts

- **Backend API:** The server's beating heart, the backend API (Application Programming Interface) orchestrates data processing, storage, and communication between the frontend and ancillary services. As the application's backbone, it ensures data management remains efficient and secure. In our project, Node.js builds the backend API, delivering a seamless, scalable solution for data flow between the frontend web app and site scanning operations.
- **Frontend Web App:** As the user-facing component, the frontend web app showcases information and facilitates user interaction with the service. Constructed with HTML, CSS, and JavaScript, the frontend relies on libraries and frameworks like Bootstrap 5, jQuery, and EJS to forge a responsive, accessible, and engaging interface.
- **Data Preprocessing:** This vital step cleans, transforms, and primes raw data for machine learning algorithms, ensuring an optimal format devoid of inconsistencies, missing values, and noise that could mar the model's performance. In our project, techniques like OneHotEncoder and SimpleImputer deftly convert categorical features into binary vectors and fill gaps in numerical features.
- **Train-test Split:** This technique partitions the dataset into distinct sets for model training and performance testing on unseen data. It's key to gauging the model's generalization prowess and staving off overfitting. In our project, an 80-20 train-test split separates the pre-processed dataset into training (80

- **Model Evaluation:** Assessing a machine learning model's performance hinges on comparing its predictions to actual outcomes. Metrics such as accuracy, precision, recall, and F1-score quantify the model's prediction accuracy. In our project, the test set evaluates the Random Forest classifier's capacity to identify malicious websites.
- **Responsive Design:** This web design strategy ensures layouts adapt to varying screen sizes and devices, delivering a consistent experience across diverse platforms. Responsive design employs fluid grids, adaptable images, and CSS media queries to craft user interfaces that automatically adjust to the user's device. In our project, the frontend web app adheres to responsive design principles to guarantee cross-device compatibility and a seamless experience.
- **AJAX:** Asynchronous JavaScript and XML (AJAX) facilitates asynchronous server requests without reloading the entire web page, enabling real-time updates and swift user interactions. AJAX leverages JavaScript and the XMLHttpRequest object for background server communication, allowing web apps to update specific page portions without a full reload. In our project, jQuery's AJAX and fetch handle asynchronous requests to the backend API and update the user interface with new data.
- **Server-Side Events (SSE):** As a web communication technology, SSE enables efficient, one-way communication from the server to the client for real-time updates without client response. SSE excels in applications requiring server updates without initiating a request, such as live data feeds, notifications, or status updates. In our project, SSE optimizes resource usage and performance during scanning, allowing backend real-time updates to the frontend without requiring client data transmission to the backend.

With a solid grasp of key terms and concepts, we can now delve deeper into the methodology.

3.2 System Architecture and Design

In this section, we unravel Sus Guru's intricate system architecture and design. We'll examine the backend API, frontend web application, and site scanning functions that synergistically forge a potent service for detecting and scrutinizing malicious websites.

3.2.1 Training Module

Training Script

The Python-written training script trains the model that discerns malicious and safe websites, employing numerous renowned libraries for tasks like data preprocessing, model training, and evaluation. Let's delve into the libraries and their roles in the training process:

1. **Data preparation:** The script processes data from two CSV files containing blacklisted and whitelisted websites. These datasets are labeled 1 for malicious (blacklist) and 0 for benign (whitelist) sites. They are then combined into a single dataset to create a balanced mix of positive and negative examples for classifier learning.

2. **Feature selection:** Some columns, like 'URL', 'Creation Date', 'Expiration Date', 'Redirects', and 'Scan Time', are excluded from the analysis as they might not contribute to the model's ability to differentiate between malicious and benign websites. The remaining columns are used as features for training the model, as shown and explained in the 5.10 section.

3. **Data preprocessing:** A column transformer preprocesses features, applying different transformations to categorical and numerical features. OneHotEncoder processes categorical features (e.g., 'TLD', 'Server Type', 'SSL Hostnames', 'Nameservers') into binary vectors, while SimpleImputer fills missing values in numerical features (e.g., 'Age in Days', 'Expires in Days') with a constant value (-1). This step ensures clean, complete, and suitable data for the model.

4. **Train-test split:** The preprocessed dataset is divided into a training set (80% of the data) and a test set (20% of the data) using the train_test_split function from scikit-learn. This split allows model performance evaluation on unseen data, providing an estimate of its generalization capabilities.

5. **Model training:** We train a Random Forest classifier on the preprocessed training data. The classifier, akin to a team of decision-makers (decision trees), collaborates for accurate and robust predictions. The classifier is trained on the preprocessed training data and consists of 100 decision trees (n_estimators=100), which are combined for more accurate and robust predictions. The random state parameter is set to 42 for result reproducibility.

6. **Model saving:** We save the trained Random Forest model as a file, enabling easy deployment and reuse in future applications or services, such as a web-based malicious website detection system. The saved file contains the trained Random Forest model and the preprocessor, saved as a pickle file using the joblib.dump function.

7. **Model evaluation:** The model's performance is evaluated on the test set by comparing predictions to actual labels. Key metrics, such as accuracy, classification report (including precision, recall, and F1-score), and confusion matrix, are calculated and printed, offering a comprehensive view of the model's effectiveness in detecting malicious websites.

By adhering to these steps and using the specified parameters, the training script effectively trains and evaluates a Random Forest model.

Secondary Training Script

The secondary script tests other models besides the Random Forest classifier and employs a similar approach to the first script, with modifications for evaluating multiple models.

The script imports several scikit-learn classifiers, including RandomForestClassifier, GradientBoostingClassifier, LogisticRegression, KNeighborsClassifier, MLPClassifier (a multi-layer perceptron neural network), and XGBClassifier from the XGBoost library. These models represent diverse algorithms that are trained and evaluated on the same dataset.

In this script, the data preprocessing step fills missing numerical values with the column mean instead of a constant value (-1). Additionally, the script calculates each trained model's size, offering insights into the trade-offs between model performance and size.

The script's main function trains and evaluates all the listed models, printing their accuracies, prediction times, model sizes, classification reports, and confusion matrices. This enables a comprehensive comparison between the models to determine the most suitable one for detecting malicious websites. By adhering to these steps and using the specified parameters, both the initial training script and the secondary script effectively train and evaluate various machine learning models, providing insights into their effectiveness in detecting malicious websites.

3.3 Web Application

3.3.1 AJAX Requests and Dynamic Content

The frontend web application uses jQuery's AJAX, and sometimes fetch, to make asynchronous requests to the backend API, updating the user interface with new information without requiring a page reload. This dynamic content loading enhances the user experience by providing real-time updates and reducing the time spent waiting for pages to load.

3.3.2 Backend

The backend API, written in Node.js, is known for its efficient performance and ability to handle numerous simultaneous connections. It serves as Sus Guru's backbone, coordinating the flow of data between the frontend web application and the site scanning functions, ensuring smooth communication and processing.

3.3.3 Frontend

The frontend web application serves as Sus Guru's primary user interface, enabling users to interact with the service and access its features. Built with Bootstrap 5, jQuery, and EJS, the frontend offers a responsive and modern design, simplifies DOM manipulation and event handling, and enables dynamic content rendering. This section delves into the various aspects of the frontend web application.

User Interface Design and Components

The frontend web application of Sus Guru features a visually appealing and user-friendly interface that is organized into components such as navigation bars, forms, buttons, and tables to enable easy navigation and interaction. The dark theme enhances the overall aesthetics and reduces eye strain for users who spend extended periods on the platform.

Responsive Design and Cross-Device Compatibility

To ensure cross-device compatibility, the frontend web application employs responsive design principles that allow the user interface to adapt to different screen sizes and devices. This feature enables users to access and interact with Sus Guru on desktops, laptops, tablets, and smartphones, providing a seamless experience regardless of the device used.

Accessibility and User Experience

The frontend web application of Sus Guru prioritizes accessibility and user experience by adhering to web accessibility standards, ensuring that users with disabilities can access and interact with the service. The platform also incorporates features like clear and concise error messages, tooltips, and user feedback mechanisms to improve the overall user experience.

By examining the architecture and components of the backend API and frontend web application, we gain a deeper understanding of Sus Guru's system design and the key considerations made to ensure an efficient, secure, and user-friendly platform for detecting and reporting malicious websites.

3.3.4 One-Page Design and Server-Side Events

The frontend web application employs a one-page design, eliminating the need to load additional pages while navigating through the site. This approach significantly improves the user experience by reducing load times and ensuring seamless interactions with the platform.

Initially, Sus Guru implemented web sockets for real-time communication between the frontend and backend during the scanning procedure. However, it was later switched to server-side events (SSE) to optimize resource usage and performance. Server-side events provided an efficient solution for Sus Guru's use case, as they allowed for unidirectional communication from the backend to the frontend without the need for the client to send data back to the backend.

As Alan Kay, the renowned computer scientist, once said:

> "Simple things should be simple, complex things should be possible."

This quote reflects Sus Guru's approach to selecting the appropriate technology for its specific needs without overcomplicating the solution.

By adopting a one-page design and server-side events, Sus Guru further enhances the efficiency and user experience of its frontend web application, providing a streamlined and optimized platform for detecting and reporting malicious websites.

3.4 Data Collection and Analysis Techniques

Sus Guru employs a diverse array of data collection and analysis techniques to gather information that contributes to the prediction algorithm used for identifying potentially malicious websites. While these techniques may not directly detect malicious patterns, they provide valuable insights that correlate with the characteristics of malicious websites. In this section, we explore these techniques and their unique roles in Sus Guru's detection process, categoring the functions we developed into their roles.

3.4.1 Domain and URL Analysis

The structure and composition of website addresses are key elements examined in domain and URL analysis techniques. we've implemented proprietary functions to facilitate these analyses.

In our system, we utilize countChars(), which computes the percentage of normal versus special characters present in domain names and URLs, as outlined in [11]. Furthermore, we also employ a function named free_tld_host(), to ascertain whether a website operates under a free top-level domain (TLD), a characteristic often associated with malicious activities.

3.4.2 Link Tracking and Analysis

The relationships between websites and their associated content are crucial components evaluated in link tracking and analysis.

For instance, our function getRedirects(), identifies and counts any redirects that a website initializes, either from HTML or Javascript. Additionally, the function countLinks(), has been implemented to discern and tally external and internal links, along with their respective statuses, i.e., whether they are broken or functional.

3.4.3 Content and Script Analysis

Sus Guru employs various techniques to analyze website content and scripts. Our Functions, such as count_blacklisted_words()[12] and scanForObfuscationPatterns() [13] examine web page con-

tent and embedded JavaScript files for the presence of blacklisted words or obfuscation patterns commonly found in malicious websites. This analysis allows Sus Guru to identify potential threats and vulnerabilities in the analyzed websites.

3.4.4 Machine Learning Model for Classification

Sus Guru uses a Random Forest classifier as its primary prediction model to determine whether a website is malicious or not. This model was chosen after testing four different classification models and achieving the highest accuracy with the Random Forest classifier. After the initial research of the most appropriate model for our use case, we chose a Random Forest as our first model, and that proved true to be the case here[14]. At the time of writing this section, the model's performance metrics are as follows:

Table 3.1: Classification Report

	Precision	Recall	F1-Score	Support
0	0.98	0.97	0.98	1765
1	0.97	0.98	0.97	1618
Average				
Macro avg	0.98	0.98	0.98	3383
Weighted avg	0.98	0.98	0.98	3383

Table 3.2: Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	1718	47
Actual 1	37	1581

Table 3.3: Accuracy

0.9751699674844813

In comparison, other models tested yielded the following accuracy results:

- XGBoost: 0.9794
- Gradient Boosting: 0.9621
- Multi-layer Perceptron: 0.9051
- Logistic Regression: 0.8812
- K-Nearest Neighbors: 0.8430

3.4.5 Website Testing and Certificate Analysis

Our function test_url() evaluates and tests websites employing Axios and Puppeteer, whilst capturing screenshots of the web pages. This practice allows our system to gather visual data about the website's content and structure for further analysis.

In addition, our function checkCertificate() authenticates the website's SSL/TLS certificate. This function checks the certificate's validity and discerns whether it's self-hosted, valid, or expired, providing us with crucial information to gauge the website's security and credibility.

3.4.6 External Services and Reputation Checks

The is_blacklisted_by_google() function checks if a domain or URL is blacklisted by Google. This provides valuable information on the website's reputation and potential risk. The getDNSRecords() function retrieves all DNS records of a domain and calculates the record count, offering insights into the website's infrastructure and complexity.

3.4.7 Metadata and Site Features

Sus Guru analyzes various metadata and site features that may correlate with malicious activities. The has_Favicon() function checks for the presence of a favicon [12], while the ads_analytics() function assesses the presence of Google Ads, Analytics, and Tags in the source code. The whoisReportWithAgeAndExpiry() function performs a WHOIS[11] scan, retrieves the domain's creation and expiry dates, and calculates the days until expiry and the domain's age. These functions contribute to building a comprehensive profile of the website, allowing for more accurate classification by the prediction model.

3.4.8 Code Validation and Script Scanning

The w3_validate() function sends the website's source code to the W3 validator to count errors, warnings, and informational messages. This helps evaluate the site's adherence to web standards and best practices. The scan_scripts() function locates all embedded and external JavaScript files loaded onto the site and scans them using the processFilteredScript() function. This function checks the Subresource Integrity (SRI) of the JavaScript files and compares them to known versions to detect potential modifications. If no SRI is found, the function checks another Content Delivery Network (CDN) to try and locate the file. It then calls the scanForObfuscationPatterns() function to identify obfuscation patterns in the JavaScript code.

By employing various data collection and analysis techniques, Sus Guru effectively gathers information that helps its machine learning model classify websites as malicious or benign. This

comprehensive approach allows Sus Guru to provide accurate and actionable insights for detecting and reporting malicious websites.

3.5 Collaboration with Law Enforcement and CERT Teams

In this section, we discuss the initial idea for collaboration between Sus Guru and law enforcement authorities as well as CERT teams. Due to time constraints and the lack of physical proof, as this is a prediction model, the implemented system provides users with options to directly mail the domain's registrar with their complaint and submit the site to Microsoft Security Intelligence and Google Safe Browsing.

3.5.1 Information Sharing and Reporting

Sus Guru recognizes the importance of sharing information on potentially malicious websites with law enforcement authorities and CERT teams. The platform gathers a wealth of data through its analysis techniques, such as domain and URL analysis, link tracking and analysis, and content and script analysis. This information can be used to identify potential threats and vulnerabilities and assist in the investigation and prosecution of cybercriminals. Sus Guru aims to collaborate with these authorities and teams in their pursuit of cyber justice by sharing information and providing insights into potentially malicious websites.

3.5.2 Host Notification and Takedown Procedures

In addition to information sharing, Sus Guru also seeks to take down malicious websites to prevent further harm to unsuspecting users. The platform initially planned to notify the hosting provider of the website's malicious activities and prompt them to take down the website. However, as this plan could not be fully realized, Sus Guru provides users with the necessary tools and options to report suspicious websites directly to the domain's registrar, Microsoft Security Intelligence, and Google Safe Browsing. This allows users to take action and contribute to the broader effort of combating malicious websites and cyber threats.

3.6 Public User Reporting and Voting System

Sus Guru offers a user-friendly reporting system that enables users to submit URLs for analysis, either through the website or the public API. This comprehensive reporting and voting system encourages user engagement in the identification and flagging of suspicious websites.

3.6.1 URL Submission and Analysis

When a user submits a URL, the system first checks the content length of the site and determines if it has been scanned previously. If not, the analysis commences. If the site has already been scanned, the system compares the content length of the old report with the new one. If the content length is significantly different, the site is scanned again as the content has changed.

The public API is free to use with a maximum of 5 requests per 10 seconds limit. It offers three routes:

- Submit a URL (`POST /api/scan`): Returns existing report IDs if the content length matches a previously scanned site or returns a new report if the content length is different or the site hasn't been scanned before.
- Get the report ID (`GET /api/report/<id>`): Returns the report results or the site's position in the scanning queue.
- Check scan status (`GET /api/queue`): Provides information on whether a site is currently being scanned and the size of the queue.

On the website, when a user submits a URL, they are connected to the server via server-side events while waiting for the analysis to complete, which takes an average of 15 seconds. The user receives updates on the progress until the result is ready.

3.6.2 User Voting

Users can cast one vote per site, choosing between labeling the site as safe or malicious. They have the flexibility to change their votes at any time. Although the initial plan was to automatically train the classification model with every even number of new verified safe and malicious reports, this feature has not yet been implemented due to time constraints. However, it's a promising future enhancement to the system.

As the site does not have user accounts, voting is based on the user's IP address, which is mentioned in the privacy policy. This approach allows for a more accessible and straightforward user experience while maintaining transparency with the users.

3.7 Accessibility and Availability

This section underlines Sus Guru's dedication to offering a public, free, and ad-free service to everyone, as well as the steps taken to assure the platform's accessibility and usability.

3.7.1 Public and Free Service

Sus Guru wants to make its service available to as many people as possible, making it a significant weapon in the battle against cybercrime. Sus Guru assures that individuals from all walks of life may benefit from its capabilities by providing a public, free, and ad-free service, creating a safer online experience for all.

Sus Guru's key purpose is to develop a user-friendly platform that is not just accessible but also devoid of irritating advertisements and monetization techniques. This commitment to an ad-free experience allows users to focus on the platform's core functionality without distractions, improving overall user experience and satisfaction.

Furthermore, Sus Guru's commitment to keeping its service free ensures that users are never asked to pay for access or make financial contributions. Sus Guru's approach not only makes it more accessible, but it also reaffirms its commitment to ensuring a safer online environment for everybody.

3.7.2 Platform Availability and Performance

In order to ensure the highest possible level of performance and availability, Sus Guru has been secured through the utilization of Cloudflare services, a prominent provider of web security and performance solutions. Cloudflare provides a variety of features that aid in the optimization of our platform, including Tiered Cache and rate limiting rules. During the previous seven-day period, the server received a total of 28.87k requests from 989 unique visitors, and experienced no downtime, thus demonstrating the platform's high level of reliability.

The implementation of Cloudflare's Tiered Cache feature is crucial in optimizing the performance of Sus Guru's website. By effectively caching static assets, the burden on the origin server is significantly reduced, thereby facilitating the swift delivery of content to end-users. By enabling the default caching level and setting a browser cache TTL of 12 hours, Sus Guru effectively balances the advantages of caching with the need to serve the latest content.

Rate limiting rules have been implemented to protect the API from possible misuse. Sus Guru has implemented a request limit of 7 per 10 seconds to efficiently handle and reduce excessive traffic. This measure aims to maintain the platform's accessibility and reliability for all users.

Apart from the performance and security features, Cloudflare offers significant analytics data that aids Sus Guru in overseeing its platform's usage and performance. The analytics offer valuable information regarding the aggregate number of requests, distinct visitors, and other pertinent metrics, which can facilitate the team's informed decision-making process concerning the continued advancement and enhancement of the platform.

In brief, Sus Guru guarantees the accessibility, reliability, and consistent performance of its platform worldwide by utilizing Cloudflare's services and enforcing rate limiting regulations. Sus Guru's dedication to delivering a seamless and optimal user experience is reinforced by the implementation of caching, rate limiting, and analytics data.

3.7.3 Raspberry Pi 4 Compatibility

Sus Guru's entire project has been meticulously developed to be hosted and run on a Raspberry Pi 4, notably the 4GB RAM variant. This compatibility ensures a low-cost, energy-efficient solution that is still widely accessible to a diverse set of users. The Raspberry Pi 4 is an amazing piece of technology that provides a tiny, low-power computing solution, making it ideal for hosting Sus Guru's system. With a quad-core ARM Cortex-A72 CPU and 4GB of RAM, this small powerhouse has more than enough computational power for the project.

One of the primary benefits of Raspberry Pi 4 compatibility is that it allows Sus Guru to operate with a PostgreSQL database server on a single device, avoiding the requirement for a specialized, high-spec hardware that many other platforms may demand. The efficient coexistence of the Sus Guru system and the database server on a single Raspberry Pi 4 demonstrates the project's efficiency, flexibility, and versatility.

Although scan times on a Raspberry Pi 4 may be 3-7 seconds longer than running on a dedicated server, this is a minor trade-off given the cost, energy, and space savings realized by using this compact computing solution. Sus Guru demonstrates its dedication to delivering a platform that is not only powerful and feature-rich, but also accessible and simple to install on a broad range of hardware configurations by preserving compatibility with the Raspberry Pi 4. Sus Guru's ability to deliver consistent performance and adapt to diverse computing environments is demonstrated by the Raspberry Pi 4's hardware specifications, further solidifying its status as an efficient, versatile, and cost-effective solution.

3.7.4 Emphasis on Free Dependencies

As a free service with no advertisements or other income sources, Sus Guru's project has been developed with a focus on utilizing free dependencies. Many interesting ideas and functions were considered but ultimately dismissed if they required costly APIs. This approach aligns with Sus Guru's goal of providing a truly accessible and free service that benefits all users without incurring additional costs.

By incorporating these elements into its methodology, Sus Guru establishes itself as a comprehensive solution for identifying and reporting malicious websites, fostering a safer and more secure internet environment for all users.

Chapter 4

Results and Analysis

In this section, we will discuss the evaluation metrics, the performance of the diverse array of models, and the insights derived from the analysis.

4.1 Models Tested, Their Working Mechanisms, and Use Cases

In this section, we will furnish an outline of the sundry models assessed in this inquiry and succinctly elucidate their operational mechanisms. These models involve Random Forest, Gradient Boosting, Logistic Regression, K-Nearest Neighbors, Multi-layer Perceptron, and XGBoost.

1. **Random Forest** Random Forest is an ensemble learning method that combines multiple decision trees to create a more accurate and robust classifier. It works by constructing several decision trees during training and outputs the class that is the mode of the classes predicted by the individual trees. Random Forest is known for its ability to handle large datasets and high-dimensional feature spaces effectively while mitigating overfitting.

    - **Use Case:** Random Forest proves particularly valuable in circumstances where data exhibits a high degree of non-linearity and intricate interactions amidst features. It is pertinent for both classification and regression tasks.
    - **Reason of Selection:** We incorporated Random Forest into our project owing to its sturdiness and ability to handle intricate feature interactions, which are frequently encountered in detecting malicious websites.

2. **Gradient Boosting** Gradient Boosting signifies another ensemble learning technique that constructs a potent learner by iteratively integrating weak learners (typically decision trees). In each iteration, a new weak learner is appended to the ensemble to minimize the residual errors of the preceding ensemble. Gradient Boosting is acknowledged for its capacity to handle both classification and regression tasks with elevated accuracy and precision.

- **Use Case:** Gradient Boosting is efficacious in circumstances where data possesses weak signals that can be fused to build a potent learner. It is particularly helpful for managing imbalanced datasets.
- **Reason of Selection:** We incorporated Gradient Boosting in our project to delve into the potential of boosting algorithms in detecting malicious websites, particularly when addressing potentially imbalanced datasets.

3. **Logistic Regression** Logistic Regression embodies a statistical technique employed for binary classification tasks. It models the likelihood of a given data point belonging to a specific class based on a logistic function. Logistic Regression is a relatively elementary model that functions optimally when the correlation between the input features and the output class is linear or nearly linear.

    - **Use Case:** Logistic Regression is befitting for binary classification tasks where the connection between input features and the output class is straightforward and linear.
    - **Reason of Selection:** We incorporated Logistic Regression in our project to act as a baseline model and to assess the assumption of linear correlations between features and the target class.

4. **K-Nearest Neighbors (KNN)** KNN represents a non-parametric, instance-based learning algorithm employed for both classification and regression tasks. It functions by calculating the distance between a new data point and all other data points in the training set, then selecting the K nearest data points to determine the class of the new data point. KNN is elementary to implement and can be effective for small to medium-sized datasets; however, its performance may deteriorate with high-dimensional feature spaces.

    - **Use Case:** KNN is suitable for small to medium-sized datasets with low-dimensional feature spaces, where the similarity between instances can be effectively measured using distance metrics.
    - **Reason of Selection:** We incorporated KNN in our project to explore the potential of instance-based learning for detecting malicious websites and to appraise its performance in comparison to other models.

5. **Multi-layer Perceptron (MLP)** MLP represents an artificial neural network comprising numerous strata of interrelated nodes or neurons. It employs a feedforward mechanism to propagate input data through the layers and backpropagation with gradient descent to update the weights during training. MLP is adept at modeling intricate, non-linear relationships between input features and output classes, rendering it apt for an extensive array of tasks.

    - **Use Case:** MLP proves fitting for tasks with intricate, non-linear relationships between input features and output classes, making it appropriate for a wide range of tasks, encompassing image and speech recognition, natural language processing, and diverse classification and regression problems.
    - **Reason of Selection:** We incorporated MLP in our project to examine the potential of artificial neural networks in detecting malicious websites and to assess its performance compared to other models, particularly when managing complex feature interactions.

6. **XGBoost** XGBoost embodies an optimized distributed gradient boosting library that executes the gradient boosting decision tree algorithm. It is designed to be highly efficient, flexible, and portable. XGBoost employs a regularized learning objective and advanced techniques to control overfitting, making it a potent and precise classifier.

    - **Use Case:** XGBoost is suitable for an extensive range of classification and regression tasks, including those with large and high-dimensional datasets. It is particularly useful for handling imbalanced datasets, as it provides options for managing class imbalance through weighting or resampling.
    - **Reason of Selection:** We incorporated XGBoost in our project due to its reputation for high performance and accuracy, as well as its ability to handle large datasets and potential class imbalance in detecting malicious websites.

By scrutinizing and comprehending the working mechanisms of these models, we can better interpret their performance within the context of malicious website detection and select the most appropriate model based on the problem's specific requirements and attributes.

4.2 Evaluation Metrics

To evaluate the effectiveness of the models, we utilized several measures, including accuracy, precision, recall, and F1-score. These metrics aid in understanding the models' efficiency in categorizing malicious and benign websites. In this section, we will examine the assessment measures employed to evaluate the models' performance in classifying malicious and benign websites. We will dissect and elucidate each metric comprehensively, utilizing the Random Forest model as an exemplar. The metrics for this model include:

1. **Accuracy (0.9710):** Accuracy represents the proportion of instances accurately classified out of the dataset's total instances. In this scenario, the Random Forest model accurately classified 97.10

2. **Precision (0.9762):** Precision is the proportion of true positive instances among instances predicted as positive. It evaluates the model's capacity to avert false alarms, i.e., classifying benign websites as malicious. A precision of 0.9762 signifies that 97.62

3. **Recall (0.9620):** Recall is the proportion of true positive instances among the actual positive instances. It evaluates the model's ability to identify malicious websites. A recall of 0.9620 signifies that the Random Forest model identified 96.20

4. **F1-score (0.9691):** The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both precision and recall, especially when there is an uneven distribution of classes. An F1-score of 0.9691 indicates that the Random Forest model achieved a good balance between precision and recall, effectively detecting malicious websites while minimizing false alarms.

5. **Prediction Time (0.1046 seconds):** This metric denotes the time required for the model to make predictions on the dataset. The Random Forest model has a prediction time of 0.1046 seconds. While not as fast as some other models, this prediction time is still reasonable for many use cases.

6. **Model Size (20.27 MB):** The model size metric signifies the storage space needed for the model. The Random Forest model has a size of 20.27 MB. While this is larger than some other models, it may still be suitable for deployment depending on the specific requirements and available resources.

These metrics offer a comprehensive view of the Random Forest model's performance in categorizing malicious and benign websites, emphasizing its strengths in accuracy, precision, recall, and F1-score.

4.3 Model Performance

We evaluated the effectiveness of various models in detecting malicious websites, including Random Forest, Gradient Boosting, Logistic Regression, K-Nearest Neighbors, Multi-layer Perceptron, and XGBoost. The performance outcomes for each model are as follows:

Table 4.1: Model Comparison

Model	Accuracy	Prediction Time	Model Size	Precision	Recall	F1-score
Random Forest	0.9710	0.1046s	20.27mb	0.9762	0.9620	0.9691
Gradient Boosting	0.9621	0.0080s	0.16mb	0.9619	0.9581	0.9600
Logistic Regression	0.8812	0.0000s	0.12mb	0.8571	0.8986	0.8773
K-Nearest Neighbors	0.8430	8.5429s	4.48mb	0.8266	0.8465	0.8365
Multi-layer Perceptron	0.9051	0.0131s	18.33mb	0.9600	0.9601	0.9600
XGBoost	0.9794	0.0095s	0.22mb	0.9856	0.9706	0.9780

By comparing the performance metrics of the various models, we can determine which models are most effective in detecting malicious websites. The XGBoost model exhibited the highest accuracy, precision, recall, and F1-score, making it a potent model for detecting malicious websites. However, the Random Forest model, with its high accuracy and reasonable prediction time and model size, was chosen for deployment due to its balance of performance and practicality.

4.4 Insights and Analysis

Based on the results, we can observe the following insights and analysis:

- Among the assessed models, the Random Forest classifier achieved the highest accuracy (0.9710) in identifying malicious websites. This implies that the ensemble learning approach

employed by Random Forest, which amalgamates multiple decision trees, is highly effective for this categorization task.

- The XGBoost and Gradient Boosting classifiers also performed admirably, with accuracies of 0.9794 and 0.9621, respectively. These models utilize gradient boosting techniques, which iteratively construct and refine weak learners to create a potent learner. This showcases the potential of boosting algorithms in detecting malicious websites.
- The Logistic Regression and K-Nearest Neighbors models exhibited relatively lower accuracies (0.8812 and 0.8430, respectively). This insinuates that these models may not be as effective in capturing the intricate patterns and relationships present in the dataset.
- The Multi-layer Perceptron, an artificial neural network, attained a commendable accuracy of 0.9051. While it did not surpass the Random Forest, XGBoost, or Gradient Boosting models, it demonstrates the potential of neural networks in this classification task.
- Regarding prediction time, Logistic Regression was the swiftest, followed by Gradient Boosting and XGBoost. While Random Forest had the highest accuracy, its prediction time was longer than other models. This trade-off between accuracy and prediction time ought to be considered when selecting a model for deployment.
- The model sizes varied, with the largest being the Random Forest model (20.27 MB) and the smallest being the Logistic Regression model (0.12 MB). Model size may be a crucial factor to consider for deployment, particularly in resource-constrained environments.

In summation, the results and analysis reveal that ensemble learning methods, such as Random Forest, Gradient Boosting, and XGBoost, are highly effective in detecting malicious websites. However, the choice of model may depend on factors like prediction time and model size, based on the specific deployment requirements. Further investigation and experimentation could concentrate on enhancing the models' performance or exploring alternative machine learning techniques and architectures to improve the detection capabilities.

4.5 Visualization of Model Performance

To better comprehend the performance of each model and facilitate comparisons, we can create visualizations that display the key evaluation metrics. In this section, we present bar plots for accuracy, precision, recall, F1-score, prediction time, and model size.

4.5.1 Accuracy Comparison

The Accuracy Comparison chart below showcases the accuracy of six distinct machine learning models. The models' competence is evaluated by their capacity to accurately discern between malicious and legitimate websites. Here's an analysis of the results depicted in the chart:

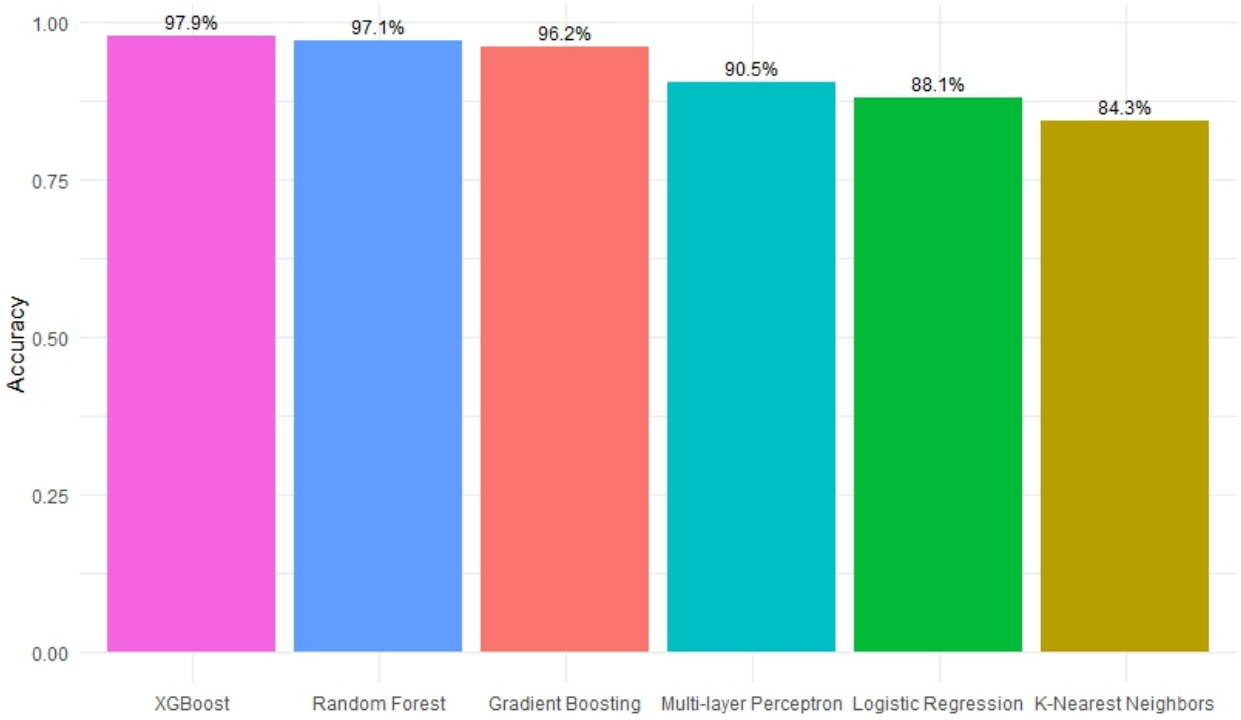

Figure 4.1: Comparison of precision among six distinct machine learning models

1. XGBoost reigns supreme with an accuracy of approximately 97.94%. This implies that XGBoost emerges as the top contender among the six models for accurately categorizing websites as malicious or benign.

2. Random Forest follows closely, claiming second place with a 97.10% accuracy, which is undeniably impressive but marginally lower than XGBoost.

3. Gradient Boosting secures the third position, boasting an accuracy of 96.21%. Although its performance is commendable, it falls short of XGBoost and Random Forest.

4. Multi-layer Perceptron occupies the fourth spot with an accuracy of 90.51%. Its precision pales in comparison to the top three models.

5. Logistic Regression ranks fifth with an accuracy of 88.12%. Despite a lower accuracy rate than its counterparts, it still delivers a satisfactory performance.

6. K-Nearest Neighbors trails at the bottom, with an accuracy of 84.30%. This outcome signifies that, among the models assessed, K-Nearest Neighbors is the least adept at predicting a website's malicious nature.

The chart unequivocally demonstrates that XGBoost, Random Forest, and Gradient Boosting outperform the others in terms of accuracy when determining a website's character. These models merit strong consideration for detecting malicious websites.

4.5.2 Precision Comparison

The Precision Comparison chart below exhibits the precision of six distinct machine learning models when categorizing websites as malicious or legitimate. Precision is a metric that gauges the proportion of true positive predictions to the overall number of positive predictions (true positives + false positives). It reflects the model's ability to accurately classify malicious websites without erroneously labeling legitimate ones as malicious. Here's an analysis of the results illustrated in the chart:

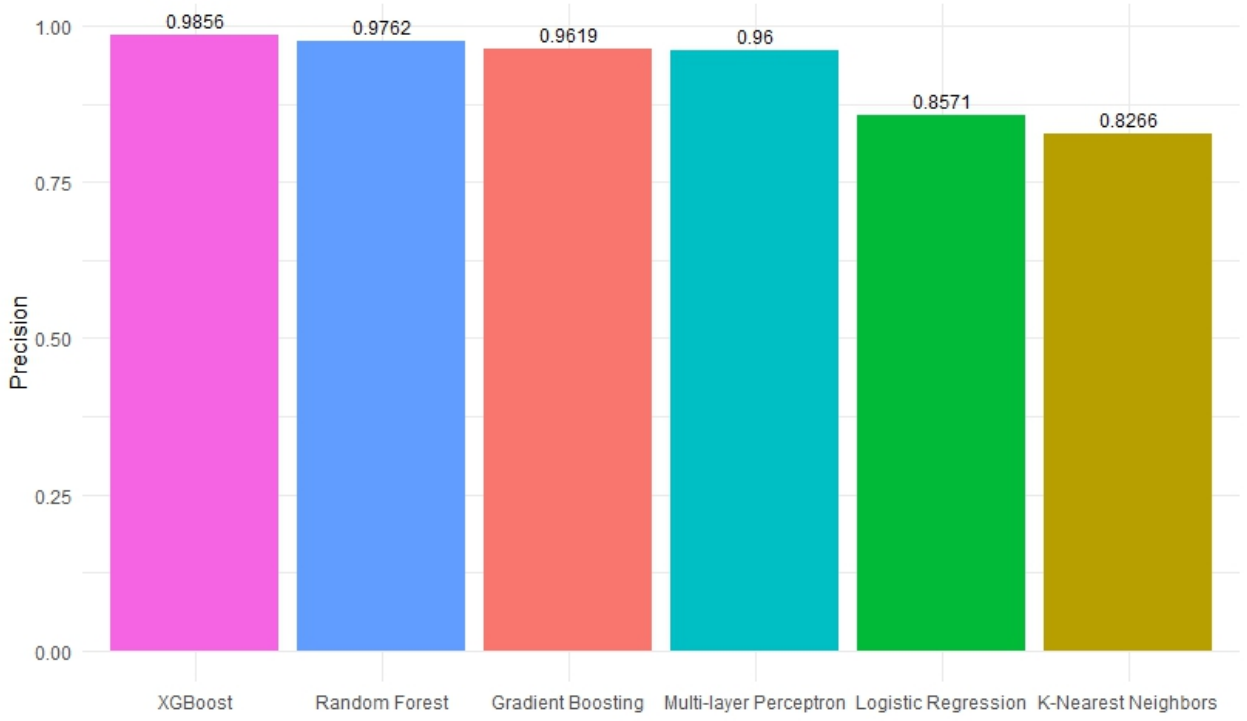

Figure 4.2: Accuracy Comparison six distinct machine learning models

1. XGBoost leads the pack with the highest precision, approximately 0.9856, signifying that it's the top model among the six for pinpointing malicious websites while minimizing false positives (legitimate websites wrongly classified as malicious).

2. Random Forest claims second place with a precision of 0.9762. Though slightly lower than XGBoost, it excels at identifying malicious websites with minimal false positives.

3. Multi-layer Perceptron secures the third position with a precision of 0.9600. Despite lower accuracy than Gradient Boosting, its superior precision implies fewer false positives.

4. Gradient Boosting ranks fourth with a precision of 0.9619. Although its precision is commendable, it falls short of Multi-layer Perceptron.

5. Logistic Regression occupies the fifth spot with a precision of 0.8571. Noticeably lower than the top four models, its precision indicates a higher likelihood of misclassifying legitimate websites as malicious.

6. K-Nearest Neighbors trails behind with the six models, at a value of 0.8266. This result indicates that the K-Nearest Neighbors model has the highest number of false positives, making it the least effective in accurately identifying malicious websites without misclassifying legitimate ones.

The chart reveals that XGBoost, Random Forest, and Multi-layer Perceptron boast the best precision when predicting a website's nature. These models should be considered top choices for detecting malicious websites while minimizing false positives.

4.5.3 Recall Comparison

The Recall Comparison chart below presents the recall of six unique machine learning models when categorizing websites as malicious or legitimate. Recall is a metric that evaluates the proportion of true positive predictions to the total number of actual positive cases (true positives + false negatives). It demonstrates the model's proficiency in detecting all malicious websites, even if it entails misclassifying some legitimate websites as malicious. Here's an analysis of the results portrayed in the chart:

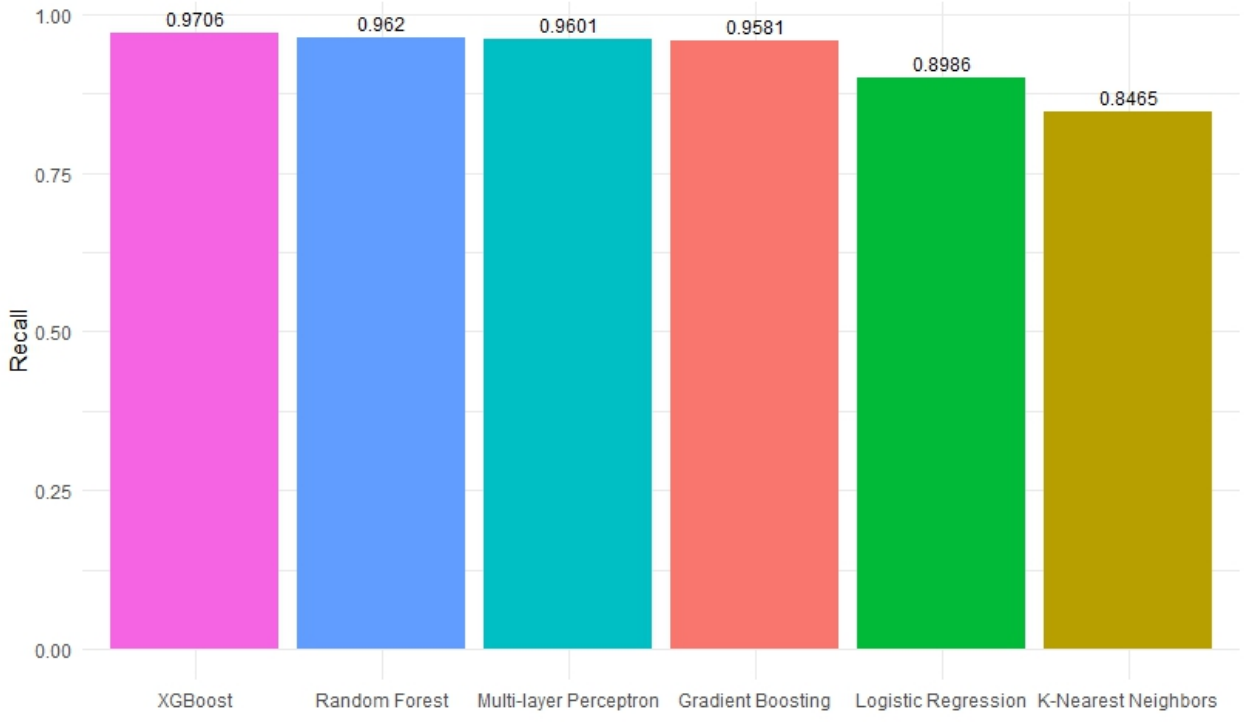

Figure 4.3: Recall Comparison of six distinct machine learning models

1. Multi-layer Perceptron takes the lead with the highest recall, approximately 0.9601, signifying that it's the top model among the six at identifying malicious websites, despite potentially misclassifying some legitimate websites as malicious.

2. XGBoost claims second place with a recall of 0.9706. Though it boasts the highest precision, its recall is marginally lower than Multi-layer Perceptron, implying that it might not identify all malicious websites as effectively as Multi-layer Perceptron.

3. Random Forest secures the third position with a recall of 0.9620. It excels at detecting malicious websites but is slightly less effective than XGBoost and Multi-layer Perceptron.

4. Gradient Boosting ranks fourth with a recall of 0.9581. Its recall is marginally lower than Random Forest, suggesting that it might overlook a few more malicious websites.

5. Logistic Regression occupies the fifth spot with a recall of 0.8986. Its recall is significantly lower than the top four models, indicating that it's less effective in identifying all malicious websites.

6. K-Nearest Neighbors trails at the bottom, with the lowest recall among the six models, at a value of 0.8465. This outcome signifies that the K-Nearest Neighbors model has the highest number of false negatives, making it the least effective in accurately detecting all malicious websites.

The chart reveals that Multi-layer Perceptron, XGBoost, and Random Forest exhibit the best recall when predicting a website's nature. These models should be considered top choices for detecting malicious websites while maximizing the identification of all malicious websites, even at the expense of some false positives.

4.5.4 F1-score Comparison

The F1-score Comparison chart below showcases the F1-score values of six distinct machine learning models when categorizing websites as malicious or legitimate. The F1-score is a metric that merges precision and recall, supplying a single value to depict the model's performance in accurately classifying malicious websites while minimizing false positives and false negatives. Ranging between 0 (worst) and 1 (best), a higher F1-score signifies superior overall performance. Here's an analysis of the chart's results:

1. XGBoost triumphs with the highest F1-score, around 0.9780, meaning that XGBoost strikes the optimal balance between precision and recall, establishing itself as the top-performing model for accurately classifying websites as malicious or legitimate.

2. Random Forest follows closely with the second-highest F1-score, at 0.9691. Although slightly lower than XGBoost, its performance remains robust, making it another suitable choice for classification tasks.

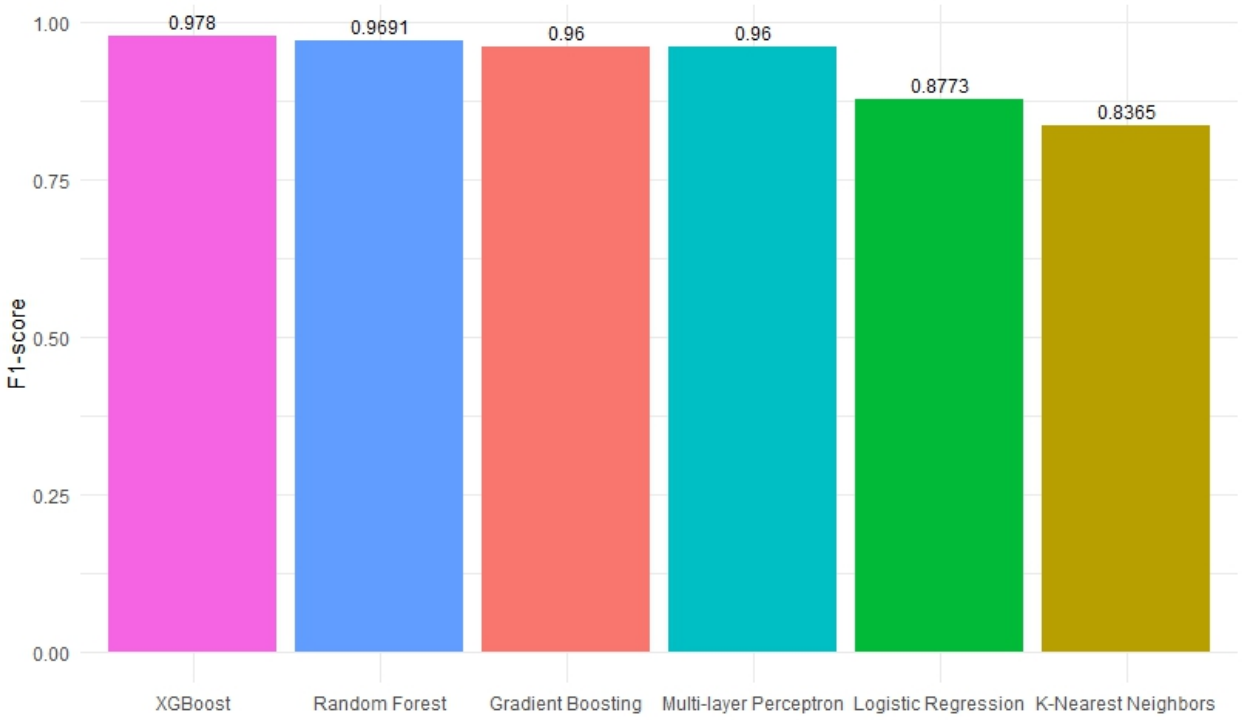

Figure 4.4: F1-score Comparison of six distinct machine learning models

3. Gradient Boosting claims third place, boasting an F1-score of 0.9600. Though its performance is somewhat lower than Random Forest and XGBoost, it still excels in balancing precision and recall.

4. Multi-layer Perceptron shares an F1-score of 0.9600 with Gradient Boosting. Despite having the highest recall, its lower precision results in a comparable F1-score to Gradient Boosting.

5. Logistic Regression ranks fifth with an F1-score of 0.8773. Its performance is considerably lower than the top four models, indicating less effectiveness in achieving a balance between precision and recall for website classification.

6. K-Nearest Neighbors lags behind with the lowest F1-score among the six models, at 0.8365. This outcome implies that K-Nearest Neighbors performs the worst in terms of balancing precision and recall when predicting a website's nature.

The chart reveals that XGBoost, Random Forest, Gradient Boosting, and Multi-layer Perceptron possess the highest F1-scores when classifying websites as malicious or legitimate. These models should be considered the prime choices for detecting malicious websites while maintaining a balance between precision and recall to minimize both false positives and false negatives.

4.5.5 Prediction Time Comparison

The Prediction Time Comparison chart below illustrates the prediction times in seconds for six distinct machine learning models when categorizing websites as malicious or legitimate. Prediction time is crucial, as it impacts a model's speed in generating results for real-world applications. Generally, a lower prediction time is preferred, indicating faster processing and more efficient resource usage. Here's an analysis of the results displayed in the chart:

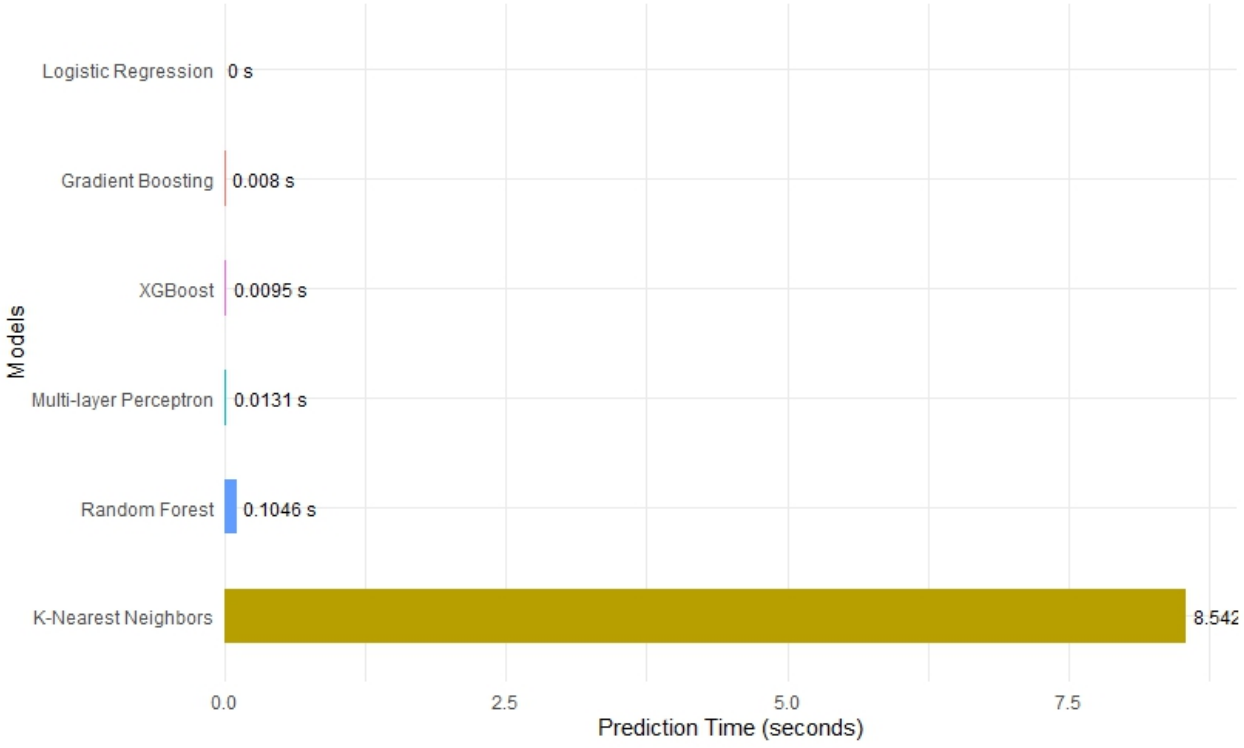

Figure 4.5: Prediction Time Comparison of six distinct machine learning models

1. Logistic Regression claims the shortest prediction time, nearly 0 seconds, marking it as the fastest model for producing website classification results.

2. Gradient Boosting follows with the second-shortest prediction time, approximately 0.0080 seconds. Though not as speedy as Logistic Regression, it remains efficient in making predictions.

3. XGBoost ranks third with a prediction time of around 0.0095 seconds. It's slightly slower than Gradient Boosting but still considered fast.

4. Random Forest has a prediction time of about 0.1046 seconds. Although slower than the previous models, it still generates predictions relatively quickly.

5. Multi-layer Perceptron features a prediction time of roughly 0.0131 seconds, faster than Random Forest but slower than the other three models.

6. K-Nearest Neighbors lags with the lengthiest prediction time among the six models, at 8.5429 seconds. This indicates that K-Nearest Neighbors is the slowest model for producing classification results, posing a significant disadvantage in real-time applications.

According to the chart, Logistic Regression, Gradient Boosting, and XGBoost boast the lowest prediction times, rendering them suitable for applications where swiftness and efficiency are vital. It's essential to weigh the trade-offs between prediction time and other performance metrics like accuracy, precision, recall, and F1-score when selecting the most fitting model for a given task.

4.5.6 Model Size Comparison

The Model Size Comparison chart below showcases the sizes of the six distinct machine learning models when trained to categorize websites as malicious or legitimate. Model size is an important aspect to consider, particularly in situations with limited memory and storage or when deploying models to edge devices. Smaller model sizes typically result in quicker loading times and reduced memory usage. Here's an analysis of the results displayed in the chart:

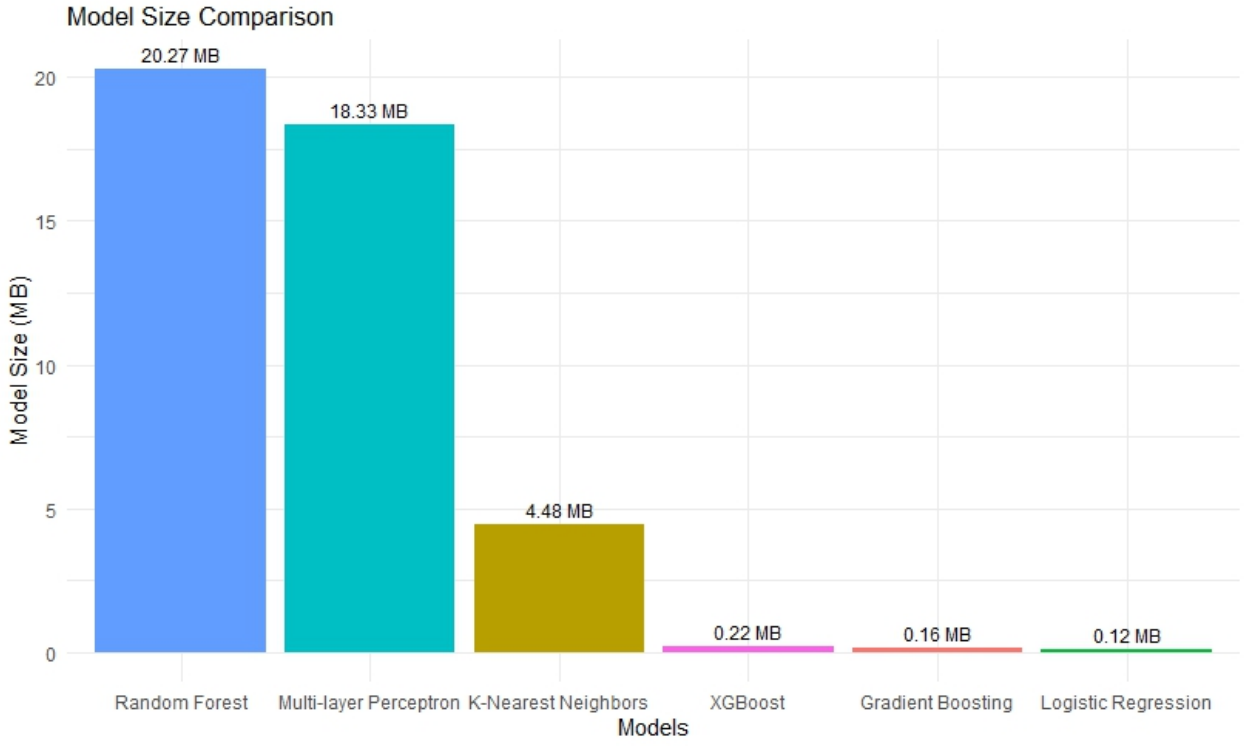

Figure 4.6: Model Size Comparison of six distinct machine learning models

1. Logistic Regression has the third-smallest model size, around 0.12 MB, indicating that it is quite memory-efficient and doesn't demand much storage space.

2. Gradient Boosting has a relatively small model size of approximately 0.16 MB, ranking it as the second smallest among the six models. This makes it a suitable choice for scenarios with limited memory and storage resources.

3. XGBoost claims the smallest model size, about 0.22 MB, rendering it the most memory-efficient model among the six. It's an excellent choice for applications with strict memory and storage constraints.

4. K-Nearest Neighbors has a model size of roughly 4.48 MB. Although larger than the previous three models, it remains relatively small compared to some of the other models.

5. Multi-layer Perceptron has a model size of around 18.33 MB. As the second-largest model among the six, it may demand more memory and storage resources than some of the other models.

6. Random Forest has the largest model size, approximately 20.27 MB, indicating that it requires the most memory and storage resources among the six models.

Based on the chart, XGBoost, Gradient Boosting, and Logistic Regression boast the smallest model sizes, making them ideal for applications with limited memory and storage. It's crucial to weigh the trade-offs between model size and other performance metrics like accuracy, precision, recall, F1-score, and prediction time when choosing the most suitable model for a given task.

4.6 Model Determination and Suggestions

Considering the outcomes, insights, and examination presented in the previous sections, we can offer suggestions for choosing the most fitting model for deployment, accounting for factors such as accuracy, prediction time, and model size, along with any specific deployment requirements or constraints. Each model has its advantages and ideal use cases, which are outlined below:

- **Random Forest:** Boasting high accuracy and robust performance, Random Forest is perfect for applications where accuracy is paramount. Its ability to manage mixed data types renders it suitable for diverse datasets. Moreover, its feature importance analysis and interpretability are invaluable for comprehending the underlying patterns in the data. Although XGBoost has marginally higher accuracy, the Random Forest model does not suffer from the issue of persistently high probabilities on false positives, rendering it more dependable in practice.

- **Gradient Boosting:** This model delivers potent performance, with accuracy slightly lower than Random Forest and XGBoost. It is suitable for applications requiring a balance between accuracy and prediction time, as it is quicker than Random Forest but not as precise. Gradient Boosting is also recognized for its ability to diminish bias in predictions.

- **Logistic Regression:** Possessing the swiftest prediction time and smallest model size, Logistic Regression is perfect for resource-constrained environments or applications where speed

is crucial. However, its accuracy is lower than the ensemble learning methods, so this trade-off should be considered. Logistic Regression is an excellent choice for linearly separable problems and offers easy interpretation.

- **K-Nearest Neighbors:** This model is beneficial in applications where proximity-based relationships are significant, but it has lower accuracy and a longer prediction time compared to other models. It may not be the best choice for malicious website detection. K-Nearest Neighbors performs well with small datasets and can adapt to changes in the data distribution over time.
- **Multi-layer Perceptron:** As an artificial neural network, this model showcases the potential of deep learning techniques in classification tasks. It provides respectable accuracy but is outperformed by Random Forest, XGBoost, and Gradient Boosting. It might be suitable for cases where non-linear relationships are prominent and larger datasets are available.
- **XGBoost:** This model offers a good balance between accuracy, prediction time, and model size. It is a strong contender for applications where both performance and resource efficiency are crucial. However, it should be noted that XGBoost consistently produced high probabilities (95%+) even on false positives, which could lead to overconfidence in its predictions and limit its practical applicability.

For our project, we chose the Random Forest classifier due to its high accuracy, robustness, ability to handle mixed data types, feature importance analysis, and interpretability. Despite XGBoost having slightly higher accuracy, the issue of consistently high probabilities on false positives makes the Random Forest model more reliable in practice. While trade-offs exist in terms of prediction time and model size, the benefits of enhanced detection capabilities and overall performance render it a suitable choice for this application.

Chapter 5

System Implementation

Embarking on an exploration of Sus Guru's system components, this chapter unravels the intricacies of algorithms, data structures, and tools deployed. Kicking off on a Raspberry Pi 4, the little beast that was used to host, analyze, and scanned approximately 10,000 websites in a row, culminating in the generation of datasets. Time restrictions and training acceleration demands prompted the rental of a Hetzner dedicated server, expediting both training and overall project momentum.

During its Raspberry Pi 4-hosted infancy, the system harnessed port forwarding, granting internet-based server access. After the port forwarding configuration was set up, a Cloudflare-proxied connection bestowed supplementary security and performance enhancements. This setup guaranteed remote server accessibility while reaping the rewards of Cloudflare's protective and optimizing services[15].

Transitioning to Hetzner's dedicated server, the Cloudflare proxy remained a constant, sustaining a secure and stable connection between server and internet. In both instances, iptables firewall configuration permitted exclusively Cloudflare's IP addresses (IPv4 and IPv6) to access the server, amplifying system security.

Throughout development, SSH protocol facilitated backend access on both platforms, ensuring secure remote entry and management of system components. Consequently, the team navigated the project with ease, implementing updates and supervising system performance, unhindered by geographical constraints.

5.1 Database Structure and Schema

The database structure and schema of Sus Guru's system are based on PostgreSQL 15.2, the most advanced version available at the time of the system's establishment. The team's commitment to continuous progress and improvement has led them to incorporate the latest PostgreSQL releases to maximize performance, strengthen security, and integrate new features. The database plays a pivotal role in managing and organizing data across various components, comprising 10 tables, 15

functions, and a single view.

PostgreSQL offers numerous advantages and applications for Sus Guru's project compared to its counterparts:

1. **Open-source:** PostgreSQL is an open-source system with a thriving, supportive community. This fosters constant growth, refinement, and an abundance of tools, libraries, and extensions, ultimately expanding the system's potential.
2. **ACID compliance:** PostgreSQL upholds ACID (Atomicity, Consistency, Isolation, Durability) compliance, ensuring reliable, consistent, and secure transactions. This guarantees data accuracy and integrity for Sus Guru.
3. **Extensibility:** PostgreSQL allows the creation of custom data types, operators, functions, and aggregates, enabling the Sus Guru team to tailor the database to their project's unique demands and facilitate bespoke functionality.
4. **Concurrency control:** PostgreSQL's multi-version concurrency control (MVCC) system permits simultaneous transaction processing without conflict, boosting performance and maintaining seamless operations, even with voluminous data or concurrent user requests.
5. **Security:** PostgreSQL offers a comprehensive suite of security features, including SSL certificates, role-based authentication, and row-level security policies, fortifying the privacy and protection of Sus Guru's stored data.
6. **Data types:** PostgreSQL supports a wide range of data types, including text, numeric, date/time, geometric, and custom, enabling the Sus Guru team to manage diverse data with utmost efficiency.

Given these advantages, PostgreSQL emerged as the optimal choice for Sus Guru's project. Its user-friendliness, compatibility with numerous programming languages and platforms, robust capabilities, performance, and adaptability assure the database's ability to meet the system's needs and contribute to the project's success.

5.2 Tables and Relationships

An overview of tables and their interrelations:

- `main`: Storing key report data, this table utilizes a unique `reportid` as its primary identifier. Fields include IP address, timestamp, URL, content length, mail ID, report time, and scan error presence.
- `score`: Holding report-specific scores, the table's primary key is the `reportid`, which refers to `main(reportid)`. It contains model predictions and probabilities.

- `website_char`: Housing URL character analyses for reports, the primary key is the `reportid`, linked to `main(reportid)`. The table features assorted URL-related fields.
- `website_external_stats`: Focusing on external website statistics in reports, the primary key is the `reportid`, connected to `main(reportid)`. Fields encompass Google-related data, Cloudflare, SSL hostnames, and w3 validation details.
- `website_metrics`: Storing website metrics for reports, the primary key is the `reportid`, referencing `main(reportid)`. Fields include redirects, links, blacklisted words, load time, server type, and more.
- `website_script_based`: Capturing script-based website elements in reports, the primary key is the `reportid`. The table features embedded scripts, external scripts, and iframes.
- `website_stats`: Encompassing general website statistics for reports, the table contains SSL data, URL shortening, free TLDs and hosting, favicons, nameservers, DNS records, and nudity indicators.
- `logs`: Logging user-system interactions, the primary key is the `id`. Fields cover timestamp, log type, user IP, status code, HTTP method, and route.
- `abuse_emails`: Storing abuse email addresses linked to reports, the primary key is the `mail_id`. The table includes an abuse email address field.

5.3 Functions

The functions were created specifically to help us in communicating with the database and transferring part of the application's logic to it. A significant advantage of employing database functions is that they can successfully avoid SQL injection attacks when implemented with appropriate permissions and backend design. This is because the user we connect to the database is not the database's administrative user. As a result, certain processes and functions in the database are defined with correct privileges and can only be run by specific users, eliminating security risks associated with potential SQL injection attacks.

Sus Guru's database comprises various functions to accomplish an array of tasks and operations. This article provides a brief, comprehensive overview of these functions, with more detailed explanations to follow:

f_delete_report(:p_reportid)

This function is used by administrators. The function executes from administrators' interfaces to delete any desired report. To execute this kind of function you have to be connected to the admins user interface. The functions needs as an argument the unique report number (reportid), which is used to execute the appropriate query inside the function and delete the specified report.

f_get_counts()

The `f_get_counts` database function, a PL/pgSQL function, yields a table with assorted counts connected to reports and model predictions. This function takes no input parameters and returns a table with columns such as:

1. `total_reports_count`: Total reports in the database.
2. `model_prediction_zero_count`: Reports with a model prediction of 0.
3. `model_prediction_one_count`: Reports with a model prediction of 1.
4. `todays_total_reports_count`: The total number of reports submitted today
5. `todays_model_prediction_zero_count`: The number of reports submitted today with a model prediction of 0.
6. `todays_model_prediction_one_count`: The number of reports submitted today with a model prediction of 1..
7. Columns like the previous two bullet points, for the last two days.

The function employs a SELECT query with multiple aggregate functions (COUNT, SUM) and conditional statements (CASE WHEN) on `main` and `score` tables. It calculates counts by joining tables on the `reportid` field and filtering records based on date truncation and model predictions.

The returned table offers an overview of reports and model predictions over the past few days.

f_get_fa_and_fn_reports()

This function is called when an admin user login to the administrator panel. What it is done with this Function? It returns the potential false positives or true negatives to the administrator, so the administrator can do more about that. It doesn't get any value as a parameter.

f_get_if_is_reported(:repid)

This function's role is to return the email from the host's or domain name provider's abuse team to us in alphanumeric format. The rationale behind this is as follows: we check to see whether there is any abuse mail linked with the report number (”reportid”) given as a parameter, and if there is any email recorded in the database with that report id, we return it to the application user.

f_get_latest_reports()

This stored method is in charge of returning the last 15 reports to us. The specific function is employed in the category projectors of the website's latest 15 reports, which we can differentiate as an option in the website's nav bar. The returned data type is a json type, which we can and do handle quite easily with node js.

f_get_mail_id(:email)

The following function above is used for updating the table that stores abuse mail. However, this is not the only task he must complete. In detail, it receives the email and, if it exists, returns the id of the items that exist without re-entering data into the table. But what if the website doesn't have any abuse mail? In this situation, the page will return 'Abuse mail not reachable' thus when it receives null as a parameter. Postgres' coalesce method is used to do this.

f_get_report(:p_reportid, :p_user_ip)

This is the mechanism that is in charge of returning to us all of the data that makes up a complete report of a web page. But how do you display a report in our app? The following information is required. The model's probability, the category in which the model categorizes the page, i.e. whether the page is good or bad, the day and time the page was requested to be checked, upvote amounts, and downvote amounts the website has, what the specific user has voted for, and the page's abuse mail. It requires two parameters: the reportid and the IP address of the user. You do not record that a report was defeated, but you utilize it to know what the preceding person voted for. When someone wants to inspect a page or votes on whether a page is dangerous or not, an IP address is recorded in the database.

f_give_a_vote(:n_user_ip, :vote, :n_reportid)

The `f_give_a_vote` database function handles user votes on reports. It takes three parameters: user's IP address (`n_user_ip`), vote value (`vote`), and report ID (`n_reportid`). The function performs actions based on the user's voting history and returns an integer indicating the result. The steps are:

1. Check for no existing user vote on the report: insert a new vote record and set the result to 0.
2. If an existing vote with a different value exists: update the votes table and set the result to 1.
3. If an existing vote with the same value exists: delete the vote record and set the result to 2.
4. If none of the above conditions are met, set the result to -1.

The function returns the result as an integer, signifying the action taken. It offers a flexible way to manage user votes, allowing users to insert, update, or delete their votes based on their previous voting history for a specific report.

f_insert_data_from_json(:reportidn, :json_data)

This function is one of the most important functions in our database. This process is in responsible for entering the primary information into the database. Which, in detail, accepts as parameters reportid and a json file. The reportid is the primary key for adding data into the attribute tables, whereas the json file contains all the metrics that the machine learning algorithm uses to determine whether a page is harmful. Apart from the features that are passed to the base in the form of json, we later asked why we needed two functions to enter characteristics and another for the score, so they were combined into one. In the same way, we also insert the model's score within the exact function.

The function proceeds as follows:

1. The initial step in this function is to add the prediction and probability values to the score table. This value is contained within the json parameter.
2. The next step is to set the values of all the parameters in each of the five parameter tables. All of these values are stored in a json file.
3. Next, add the abuse team's email address and id to the appropriate tables.
4. At the end of the function, if no errors are encountered, the function returns the values required for a site report.

f_just_email_reported(:repid)

The `f_just_email_reported` database function updates the `time_reported` field of a specific report in the `main` table. It takes the report ID as input and sets the `time_reported` field to the current local timestamp, returning 1 for a successful update.

f_maininserter(:ipadd, :cl, :checkurl)

The `f_maininserter` function is a PostgreSQL stored function in PL/pgSQL that inserts a new report into the `main` table or fetches existing reports based on input parameters. It takes three inputs: IP address (`ipadd`), content length (`cl`), and URL (`checkurl`).

The function proceeds as follows:

1. Declaration of variables: `existing_reports`, `report_id`, `result`.

2. Fetch existing reports by performing a SELECT query on the `main` table.
3. Check for existing reports:
    - If existing reports are present (`cardinality(existing_reports) > 0`), the function constructs a JSONB object with a key `'existing_reports'` and the `existing_reports` array as its value, storing this in the `result` variable.
    - If no existing reports are found, the function generates a new report ID (`report_id`) using the `seq_repid` sequence. It then inserts a new row into the `main` table with the given IP address, report ID, timestamp, URL, and content length. Lastly, it constructs a JSONB object with a key `'new_report'` and the `report_id` as its value, storing this in the `result` variable.
4. Return the result: The function returns the `result` JSONB object, which will either contain the `'existing_reports'` key with an array of existing reports or the `'new_report'` key with the new report ID.

In summary, the `f_maininserter` function verifies the existence of reports in the database with the same URL and content length. If any are identified, it returns them as a JSONB object; otherwise, it generates a new report in the database and returns the new report ID as a JSONB object.

`f_set_report_error(:p_report_id, :p_scan_error)`

The `f_set_report_error` database function, a PL/pgSQL function, updates the `scan_error` field of a specific report in the `main` table. It takes the report ID and scan error status as input parameters and updates the corresponding record. The function doesn't return any value, as its sole purpose is to execute the update.

The function can be called using the following syntax:

SELECT f_set_report_error(:p_report_id, :p_scan_error);

where `:p_report_id` and `:p_scan_error` are input parameters that need to be replaced with appropriate values. The function updates the `scan_error` field of the report identified by `:p_report_id` with the value of `:p_scan_error`.

In summary, the `f_set_report_error` function is a simple PL/pgSQL function that updates a specific report's scan error status. It's commonly used in error handling and reporting scenarios.

f_just_email_reported

The `f_just_email_reported` database function updates the `time_reported` field of a specific report in the `main` table. It takes the report ID as input and sets the `time_reported` field to the current local timestamp, returning 1 for a successful update.

5.4 The score at the database

Our first thought is that after we pass all the website features to the database, a second score may be generated using the characteristic data that we save in the database. But how would such a thing be put into action? The plan was to construct a view that, for each numerical value, would divide the average of the table's values into two rows, one for good pages and the other for bad pages. The benefit of views is that they are automatically and rapidly refreshed with new values that may be inserted into the attributes tables mentioned at 5.2 Tables and Relationships.

What about Boolean properties? We could not take an average of what is true and false because of their nature, which can take two states, it is quite easy for a website to be an exception and create false positives / false negatives. This term refers to when a website is bad and when it is good, and vice versa. We used the training sets we built to train the machine learning model to extract statistics and combinatorial items with a high percentage of malicious pages.

So, initially, two functions were created: the first function fetched an attribute and a value from the table and checked how much the deviation of values had between them. If the deviation was large between bad and good pages at a percentage of more than 60%, the attribute was considered good. If the entered attribute belongs to the bad subset, 1 is returned otherwise, 0 is returned, -1 is returned if no good attribute is observed. After completion of the process for all the features, we calculated the percentage of features worth working on, which was 40% of the final score.

The remaining 60% was derived from the combination of attributes. This method of scoring did not work well. Various tests were performed, but the percentages were discouraging. A typical example is a malicious page that scored 40 percent bad data and our safe pages that provided 15-20 percent good data. We chose not to use this type of score after considerable consideration and many percentages testing cause it would detract from the value supplied by the established machine learning model. Although it was quite interesting. There is hope that the application will be upgraded in the future. Perhaps in the upgraded version if it has no effect on the score, but it's like a way to find false positives and false negatives.

5.5 Backend Algorithms and Implementation

5.5.1 Site Scanning Algorithm

The backend algorithm for website scanning is incorporated in the "/scan" POST route. When a client sends a request to this endpoint with a URL for scanning, the subsequent steps occur:

1. Validate input:
   (a) Assess URL presence and validity. If absent or invalid, return an error.
   (b) Evaluate if the URL is an IP logger. If true, return an error.

   (c) Determine if the URL directs to a file by checking file extensions. If affirmative, return an error.

2. Examine the scanning queue status for the given report ID:

   (a) If the report is being scanned or in the queue, return the queue status.

3. Obtain user IP address and log it.

4. Invoke the ‘test_url‘ function with the URL and browser instance.

   (a) The function fetches source code, headers, status code, and load time for the URL.

   (b) In case of an error, return an error message to the client.

5. Calculate the content length of the source code.

6. Call the ‘mainInserter‘ function with user IP, content length, and URL.

   (a) This function invokes the ‘f_maininserter‘ database function that fetches existing reports or inserts a new report into the database.

7. If existing reports are discovered, return them to the client.

8. If a new report is generated, add it to the scanning queue and return the new report ID to the client.

The ‘test_url‘ function, as mentioned earlier, is responsible for fetching website information such as source code, headers, status code, and load time. It uses the Puppeteer library to interact with the website in a headless browser environment, allowing it to mimic real user interactions and obtain accurate results.

The ‘mainInserter‘ function is a wrapper for calling the ‘f_maininserter‘ database function. It takes the user IP, content length, and URL as input parameters, and then returns either the existing reports or the new report ID based on the database function’s result.

In summary, the backend scanning algorithm consists of validating the input URL, checking its queue status, fetching website information, checking for existing reports or creating a new one, and updating the scanning queue. The whole process is performed asynchronously, ensuring efficient and non-blocking operation.

5.5.2 Site Get Report Algorithm

The algorithm for the ”/report/:id” GET route serves the purpose of fetching a specific report by its ID. When a client sends a request to this endpoint, the following steps are performed:

1. Extract the report ID from the request parameters.

2. Get the user's IP address and log it.
3. Check the scanning queue status for the given report ID using the 'checkQueueStatus_id' function:
   (a) If the report is currently being scanned or in the queue, return the queue status to the client.
4. If the report is not in the queue, call the 'get_report' function with the report ID and user IP to fetch the report from the database.
   (a) If the report is not found, return an error to the client.
   (b) If the report has an error flag, return an error indicating the website could not be scanned.
5. If the report is found and does not have an error flag:
   (a) Generate the image URL using the 'generateHash' function.
   (b) Add a random message from the 'scan_messages' array to the report object.
6. Return the report to the client with a 200 status.

The algorithm is responsible for handling the client's request for a specific report. It checks whether the report is being scanned or in the queue,

5.6 Tools and Technologies

In this segment, we'll delve into the primary tools and technologies employed for Sus Guru's system creation, management, and performance, exploring their applications, advantages, and reasons for selection.

1. **Visual Studio Code**: A versatile and robust integrated development environment (IDE) by Microsoft, Visual Studio Code (VSCode) caters to Sus Guru's diverse needs with its support for numerous programming languages, libraries, and frameworks. Its flexibility, efficiency, and ease of use, combined with features like syntax highlighting, code completion, debugging, version control integration, and a vast array of extensions, make VSCode the ideal choice for the project.
2. **DBeaver**: This open-source, cross-platform SQL client and database management tool, compatible with various databases, was employed to manage and interact with Sus Guru's PostgreSQL database. DBeaver's visual query building, data import/export, ERD generation, and database schema management features contribute to its selection for compatibility, comprehensive features, user-friendliness, and cost-free use.

3. **Thunder Client**: An easy-to-use REST API testing tool integrated into VSCode as an extension, Thunder Client supports numerous HTTP methods and authentication mechanisms. With features like environment variables, request history, and import/export of requests, it was employed for testing API routes during Sus Guru's development.

4. **Adobe Photoshop**: As an industry-leading image manipulation and graphic design software, Photoshop provides comprehensive tools for creating, editing, and enhancing visuals. Sus Guru employed Photoshop for designing visuals like logos, banners, and UI elements, capitalizing on its powerful capabilities, precision, and creative freedom.

5. **Cloudflare Panel**: As a global CDN, web infrastructure, and security provider, Cloudflare offers services such as DNS settings, SSL/TLS encryption, caching, and rate limiting. Sus Guru utilized the Cloudflare Panel to manage DNS settings and implement rate limiting, ensuring optimal performance, availability, and security.

6. **RStudio**: An R programming language IDE, RStudio streamlines development processes and facilitates the creation of high-quality graphs and visualizations. Sus Guru harnessed RStudio to generate informative graphs for the scoring system and paper, providing valuable insights and visual representations.

7. **Discord**: This versatile communication platform enabled seamless communication between Sus Guru team members through voice, video, and text chat, ensuring efficient collaboration and information sharing, contributing to cohesive development and effective decision-making.

8. **GitHub**: A widely used web-based platform for version control and collaboration, GitHub provided Sus Guru with project management tools and helped manage source code, track changes, and collaborate on the system development, ensuring an organized development process and efficient teamwork.

9. **Zoom**: A popular video conferencing and communication tool, Zoom facilitated real-time communication between the Sus Guru team and their professor, ensuring the project stayed on track and received valuable input and feedback.

These tools and technologies were carefully chosen to ensure a robust, efficient, and seamless development experience, resulting in a high-performing, secure, and user-friendly system. Their combination facilitated a streamlined workflow and contributed to Sus Guru's platform's successful creation.

5.7 Libraries used in the Backend

In this section, we'll examine the libraries employed in the backend, their applications, and their contributions to the project's overall functionality. These libraries furnish essential tools for processing, analyzing, and manipulating data required for the scanning process.

1. **dotenv**: This package securely manages sensitive data, like API keys or credentials, by loading environment variables from a ‘.env‘ file into the ‘process.env‘ object. In this project, dotenv handles sensitive information.

2. **http and https**: These built-in Node.js modules manage HTTP and HTTPS requests, respectively, providing low-level APIs for web server and service interactions. In this project, they handle tasks like fetching website data and API responses.

3. **cheerio**: This library offers a jQuery-like syntax for parsing and manipulating HTML documents, enabling efficient information extraction from web pages' source code. In this project, Cheerio parses and extracts data during the scanning process.

4. **whoiser**: A WHOIS client for Node.js, whoiser retrieves domain information, such as the domain owner, registration date, and expiration date. In this project, whoiser collects WHOIS data for analyzed websites.

5. **axios**: A popular library for making HTTP requests in Node.js and browser environments, axios offers a promise-based API, simplifying asynchronous request and response handling. In this project, axios fetches web pages and their source code for further analysis.

6. **dns (@layered/dns-records)**: This library queries DNS records, such as A, AAAA, MX, or TXT records. In this project, it gathers information about analyzed websites' DNS configurations, helping understand the website's infrastructure and network setup.

7. **nodeDns (dns)**: This built-in Node.js module provides an asynchronous network wrapper for DNS resolution tasks. In this project, nodeDns queries DNS records and performs reverse DNS lookups when necessary.

8. **node-fetch**: A light-weight module that brings the ‘fetch()‘ API from browsers to Node.js environments, node-fetch offers an easy-to-use, promise-based API for making requests and handling responses. In this project, it can be an alternative to axios for fetching web pages and their source code.

9. **crypto**: This built-in Node.js module provides cryptographic functionality for hashing, signing, verifying, and encrypting data. In this project, the crypto module handles tasks requiring secure data handling or storage, such as hashing or encrypting data before database storage.

10. **tld-extract**: This library extracts the top-level domain (TLD), domain, and subdomains from a given URL. In this project, tld-extract parses URLs and gathers domain structure information, helping understand the organization of analyzed websites.

11. **child_process (spawn)**: The ‘child_process‘ module, a built-in Node.js module, allows running separate system processes. The ‘spawn‘ function creates new processes and executes external commands. In this project, it runs the model prediction script in python and retrieves the result.

12. **puppeteer-extra**: An extension for the Puppeteer library, puppeteer-extra offers additional functionality and plugins. Puppeteer, a headless browser automation library, is used for tasks

like web scraping, generating screenshots or PDFs, and testing web applications. In this project, puppeteer-extra handles advanced web scraping tasks requiring JavaScript rendering or programmatic web page interactions.

13. **puppeteer-extra-plugin-stealth**: A plugin for Puppeteer-extra that makes the headless browser resemble a regular browser, helping avoid anti-bot measure detection. In this project, the StealthPlugin, combined with Puppeteer-extra, ensures the web scraping process isn't blocked by websites detecting automated browsers.

14. **puppeteer-extra-plugin-recaptcha**: A plugin for Puppeteer-extra that automatically solves Google reCAPTCHA challenges using external services like 2captcha or Anti-Captcha. In this project, the RecaptchaPlugin bypasses reCAPTCHA challenges that may be encountered while accessing or interacting with certain websites during the scanning process. This ensures a smooth and uninterrupted data collection process.

These libraries and modules play critical roles in the project's backend, providing necessary tools and functionalities for various tasks, such as web scraping, data extraction, and analysis. By utilizing these libraries, the project can efficiently gather and process data from different sources to create a comprehensive analysis of the target websites. The combination of these libraries helps to maintain a high level of technical complexity while preserving simplicity, ensuring that the backend remains efficient and accessible for developers working on the project.

5.8 Backend Functions for Scanning Process

The backend functions serve as the linchpin in the scanning process, delving into various website aspects and producing valuable data for the system. Here's an outline of the principal backend functions employed in the scanning process, along with their respective descriptions:

1. **is_url_shortened(url_parts)**: Assessing whether a provided hostname is a URL shortener, this function employs the url_shorteners list derived from the lists/url_shorteners file. It takes an object with the URL parts and yields a boolean value signifying whether the hostname is a URL shortener. No external libraries are used.

2. **getRedirects(url, browser)**: This function recovers the redirect URLs and count for a specified URL via the Puppeteer library. Accepting the URL and a Puppeteer browser instance as parameters, the function tracks redirects by listening for the 'framenavigated' event, intercepting link clicks, and implementing a navigation timeout for JavaScript redirects. The function then returns an object containing the visited URLs and redirect count. The main library used is Puppeteer, renowned for web scraping and browser automation tasks.

3. **test_url(url_parts, browser, screenshot)**: Utilizing Axios and Puppeteer libraries, this function tests a URL. It takes an object containing the URL and domain, a Puppeteer browser

instance, and a boolean value indicating the need for a screenshot. Initially, the function attempts to fetch the URL using Axios with custom headers and a timeout. If unsuccessful or if a screenshot is requested, the function employs the Puppeteer library to obtain the source code, headers, status code, and load time. Subsequently, the function checks for the "Just a moment..." title, indicative of a Cloudflare bot block, and returns an object containing pertinent information. The main libraries used are Axios for HTTP requests and Puppeteer for browser automation tasks.

4. **checkCertificate(url_parts, timeout = 10000, retryWithoutSub = true)**: This function examines the SSL certificate for a given domain. Accepting an object with the URL parts, a timeout (default: 10,000 ms), and a boolean indicating whether to retry without the subdomain if the initial request fails, the function utilizes Node.js built-in modules http and https to request the domain, obtaining SSL certificate details from the response. It then processes the certificate information to determine if the domain has SSL, whether it's valid, if it's self-signed, and the SSL hostnames. The function returns a promise that resolves with an object containing the certificate details. The built-in Node.js modules http and https are the main libraries used.

5. **cloudflare_check(source_code, headers, status_code)**: Checking if a website is behind Cloudflare and if a bot is blocked, this function accepts the website's source code, headers, and status code as parameters. The function employs regular expressions to test for Cloudflare challenge scripts in the source code. It then checks if the website is behind Cloudflare by examining the 'server' header value and determines if a bot is blocked by assessing the status code and presence of challenge scripts. The function returns a promise that resolves with an object containing Cloudflare check results.

6. **is_blacklisted_by_google(url_parts)**: This function verifies if a URL is blacklisted by Google Safe Browsing. It takes an object containing the URL parts as input. The function constructs a request body that includes the client and threat information and sends a POST request to the Google Safe Browsing API. The response is checked for matches, and the function returns true if the URL is blacklisted, otherwise false.

   (a) The main library used in this function is fetch for making HTTP requests to the Google Safe Browsing API.

7. g**etDNSRecords(domain)**: Retrieving DNS records for a specified domain, this function accepts a domain string as input. The function initializes an object containing record types and their counts. It then defines two helper functions: getRecordsWithRetry and getNameservers. The former is employed to fetch DNS records for a specified domain and record type, with a maximum number of retries. The latter is used to obtain nameservers for a given domain.

   (a) The main part of the function first retrieves nameservers using the getNameservers helper function, then iterates over the record types and populates the counts in the recordTypes object using the getRecordsWithRetry helper function. The function returns a promise that resolves with the DNS records.

   (b) The primary libraries used in this function are the dns module for fetching DNS records and the built-in Node.js dns module (nodeDns) for resolving nameservers.

8. **processConcurrent(tasks, handler, concurrency)**: This function orchestrates an array of tasks in parallel, adhering to a maximum number of concurrent tasks. Accepting an array of tasks, an async handler function, and a maximum concurrency level, it manages a set of active tasks and processes them with the given handler. The function awaits the completion of active tasks and returns a promise resolving to an array of results.

9. **count_blacklisted_words(source_code)**: This function enumerates the blacklisted terms within provided HTML content. Given a webpage's source code as input, the function utilizes Cheerio to load the code, extract body content, eliminate HTML tags and excessive spaces, and convert text to lowercase. Subsequently, it computes the frequency of blacklisted terms in the content and returns the percentage of blacklisted terms as a rounded figure.

    (a) The primary library employed in this function is 'cheerio' for parsing and manipulating HTML content.

10. **countLinks(url_parts, source_code)** [16]: This function quantifies links within a website's source code, taking an object containing URL parts and the website's source code as inputs. Initializing arrays for diverse link types, it extracts all links from the source code, categorizing them into domain links, short-path domain links, empty-path domain links, functional external links, and non-functional external links. The function processes external links concurrently, distinguishing between functional and non-functional links using the 'processConcurrent' function, and returns an object containing each link type's count.

    (a) The main libraries harnessed in this function are 'axios' for HTTP request execution to verify external links, and the previously defined 'processConcurrent' function for concurrent external link processing.

11. **whoisReportWithAgeAndExpiry(url_parts)**: This code outlines an asynchronous function that produces a WHOIS report for a given domain. Accepting one parameter, 'url_parts', an object containing URL components, the function returns a promise resolving to an object featuring the WHOIS report, encompassing domain creation and expiration dates, days until domain expiration, and days since domain creation. If accessible, the registrar abuse contact email is incorporated in the report.

    (a) The function first initializes several variables and defines an asynchronous 'performWhois' function to conduct the WHOIS lookup. It iterates through a series of retries to guarantee accurate WHOIS information retrieval.

    (b) Upon obtaining the WHOIS information, the function extracts creation and expiration dates and the registrar abuse contact email. The function verifies the retrieved dates' validity and converts them to ISO strings if necessary.

    (c) The 'calculateAgeAndExpiresInDays' function calculates the domain's age and days until expiration. Finally, the function returns an object containing the computed data.

12. **countChars(url)**: This function scrutinizes a given URL by calculating the total character count and the percentage of specific characters in different URL segments, such as the hostname and the remainder of the URL. First, the function creates a URL object and extracts

the hostname and remaining URL (excluding the origin). It then computes the total character count and tallies special characters in each segment. Ultimately, it calculates the percentage of specific characters (e.g., hash, question mark, ampersand, etc.) throughout the URL. The function returns a promise resolving to an object containing these character counts and percentages.

13. **free_tld_host (url_parts)**: This function examines if a given domain's top-level domain (TLD) is free. Accepting an object containing URL components (e.g., TLD, domain, subdomain), it contrasts the TLD against a predefined list of free TLDs (free_domain_tlds). Additionally, the function verifies if the domain is hosted on a free hosting service by comparing the domain against a predefined list of free hosts (free_hosts). The function returns a promise resolving to an object containing two boolean values: is_free_tld (indicating if the TLD is free) and is_free_host (indicating if the domain is hosted on a free hosting service).

14. **has_Favicon (url, source_code)** [17]: This function determines favicon existence by examining the URL and website's source code. Employing regular expressions, it searches for a favicon link in the source code. If discovered, the function returns true; if not, it attempts to fetch the favicon by creating a URL with the domain and appending /favicon.ico. A status code of 200 signifies favicon presence, returning true; otherwise, it returns false.

15. **ads_analytics(source_code, headers)**: This function inspects the presence of Google Ads, Analytics, and Tags in a website's source code and headers. Utilizing multiple regular expressions, it looks for patterns linked to Google Adsense, Analytics, and Google Tags (Gtag) in the source code. Boolean variables hasAdsense, hasAnalytics, and hasGtag are set based on the respective pattern presence. The function then returns a promise resolving with an object containing these boolean values, indicating each service's presence in the source code and headers.

16. **Scan_optimized(url, browser, test_url_object)**: Acting as the central connector, this asynchronous function unifies various elements in the analysis process. It accepts three parameters: 'url', 'browser', and 'test_url_object'. The function scrutinizes a given URL, generating a comprehensive report containing diverse information about the URL, its content, security, and more. The function executes the following actions:

    (a) Gains URL parts and inspects if the URL is shortened.

    (b) Retrieves any associated redirects with the URL.

    (c) Assesses the URL, amassing data about its source code, headers, status code, and load time.

    (d) Inspects the SSL certificate for the URL.

    (e) Verifies if the URL resides behind Cloudflare and whether its content is blocked.

    (f) Performs multiple tasks concurrently using Promise.all, such as examining if the URL is blacklisted by Google, acquiring DNS records, tallying links, detecting nudity, enumerating blacklisted words, producing a WHOIS report, calculating characters, ascertaining if it's a free TLD or host, probing for favicons, scrutinizing ads, validating the URL with W3C, and scanning scripts.

(g) Processes the results, fuses them into a singular report object, and computes supplementary information based on the assembled data.

Upon providing a 'test_url_object', the function employs a machine learning model to create a prediction, appending the result and probability to the report object.

Ultimately, the function returns the generated report object, replete with detailed information about the specified URL.

In conclusion, the Backend Functions for Scanning Process consist of several functions that work together to analyze different aspects of a website, such as the URL structure, presence of a favicon, and the use of Google Adsense, Analytics, and Tags. Collectively, these functions provide a comprehensive analysis by gathering 84 parameters that are crucial for the data needed for the model. By extracting and processing this valuable information, the model can better understand the website's characteristics, enabling it to make more accurate predictions and assessments.

5.9 Backend Functions for the API

API backend functions are essential in overseeing the scanning process, tracking the scanning queue, managing connections, and digesting data. Let's delve into each function and its role:

Here's a deeper look at each function:

1. **isCorrectUrl(url)**: A nifty utility function, isCorrectUrl harnesses JavaScript's built-in URL constructor to assess if an input string is a valid URL. If the constructor avoids error-throwing, it returns true, signifying a valid URL. In case of an error, it returns false.

2. **initializeBrowser()**: This function readies a Puppeteer browser instance armed with stealth and reCAPTCHA solving powers. Puppeteer, a Node.js library, presents a high-level API to control headless Chrome or Chromium browsers via the DevTools Protocol. The StealthPlugin conceals browser attributes, making the browser resemble a regular user agent, lowering bot-detection risks. The RecaptchaPlugin assists in overcoming reCAPTCHA challenges using the 2captcha service. The function returns the prepped browser instance for additional web scraping duties.

3. **checkQueueStatus_id(id)** & **checkQueueStatus_url(url)**: These utility functions inspect a report's status in the scanning queue, based on report ID or URL. They return an object with 'scanning' and 'queue_position' properties. The boolean 'scanning' property reveals if the report is being scanned or waiting in the queue. If true, 'queue_position' shows the report's place in line. These functions are handy for providing real-time status updates to users, like the report's progress in scanning.

4. **processQueue(sharedVariables)**: At the core of the scanning process, this function tackles the scanning queue. Its responsibilities include:

(a) Traversing the scanning queue when items require processing.

(b) Initiating a Puppeteer browser instance, if not already active.

(c) Refreshing the scanned item and launching the scan_optimized function for the scanning process, which involves visiting the target URL, extracting relevant data, and executing checks.

(d) Sending updates to the client via Server-Sent Events (SSE) through sharedVariables.connections during scanning.

(e) Once scanning is complete, inserting scan data into the database and sending a "Finished" message with the report data to the client.

(f) If errors occur during scanning, the function records the report error in the database and sends an "Error during scanning" message to the client.

By examining each function, we gain insight into the vital aspects of the scanning process, queue management, and the interplay between backend functions and the front-end, ensuring a seamless user experience.

5.10 Features

These attributes contribute to the prediction model's efficacy in discerning diverse aspects of a website, potentially signifying its legitimacy or maliciousness. This section offers a brief outline of every feature employed in the analysis of websites.

1. **URL** and **TLD**: Investigating the URL and TLD may unveil malicious website patterns, like employing specific TLDs correlated with malevolent activities.

2. **Google Blacklisted**: A website blacklisted by Google likely harbors malicious or harmful content.

3. **Server Type**: Certain server types prevalent among malicious websites can aid the model in prediction.

4. **Content Length**: Malicious sites may exhibit significantly shorter or longer content compared to typical sites, serving as a valuable predictive feature.

5. **Has SSL**, **SSL Valid**, **SSL is Self Signed**, **SSL Hostnames**, and **SSL Hostnames Count**: Analyzing SSL/TLS certificate attributes can signal a website's security and trustworthiness, as malicious sites may use self-signed or invalid certificates.

6. **Link Ends in File Extension**, **Redirects**[18], **Number of Redirects**, and **Redirect URLs Contain Port**: Malicious sites often employ redirection tactics and URLs terminating in file extensions to obscure their true intentions.

7. **Creation Date**, **Expiration Date**, **Expires in Days**, and **Age in Days** [19, 16] : Malicious sites might exhibit brief lifespans or peculiar registration patterns, serving as useful prediction features.

8. **URL Shortened** [16], **Free TLD**, and **Free Host** [12]: Malicious sites may utilize URL shortening, free TLDs, or free hosting services to decrease expenses and elude detection.

9. **Contains Iframe** [20], **Embedded Scripts**, **Embedded Obfuscated Scripts** [13], **External Scripts**, **External Obfuscated Scripts**, and **External Verified Scripts**: The existence of iframes, assorted script types, and obscured code can indicate potentially malicious content or behavior.

10. **URL Total Chars** [17] , **Hostname Total Chars** [19, 17], **Hostname Special Chars** [17], **Hostname Percentage Chars**, **Rest Total Chars**, **Rest Special Chars**, **Rest Percentage Chars**, and diverse **URL Percentage** attributes: Examining the URL and its components may reveal structural patterns in malicious websites.

11. **Scan Time**: The duration of the website scan can also serve as a useful feature, as malicious sites might employ techniques to impede or obstruct scanning attempts.

Collectively, these features bolster the prediction model's capacity to distinguish between legitimate and malicious websites. By recognizing patterns and correlations between these attributes, the model can generate accurate predictions about a website's nature.

Chapter 6

User Interface Design

Figure 6.1: Screenshot of Sus Guru's on multiple devices

The foremost objective of Sus Guru's user interface construction is to offer a user-centric, instinctive, and accessible experience for users across an array of devices and browsers. The user interface was developed employing the Bootstrap framework, vanilla JavaScript, and a few custom CSS components to establish an eye-catching, adaptable, and effortless-to-use platform.

This section will elucidate the design principles, layout, and functionality of the user interface, emphasizing its user-oriented methodology and meticulousness. Screenshots and diagrams will be supplied to demonstrate the user interface construction and its diverse features.

6.1 Introduction to the User Interface

Sus Guru's interface accentuates trio core facets: user amiability, browser congruity, and gadget adaptability. Harnessing Bootstrap framework, vanilla JavaScript, and bespoke CSS constituents, the interface achieves these goals sans sacrificing visual allure and performance.

Vanilla JavaScript, chosen for its unpretentiousness and comprehensive browser backing, enables efficient evolution without added libraries. This ensures the site remains featherweight and operates effectively across manifold devices and browsers.

Bootstrap, a prominent CSS, JavaScript framework, was preferred for its simplicity and intrinsic responsiveness, facilitating a uniform, visually captivating design adaptable to diverse screen sizes and gadgets. Additionally, Bootstrap presents a gamut of pre-constructed elements expediting development and sustaining a cohesive visual style throughout.

The site's efficacy, assessed using Google's PageSpeed Insights and Cloudflare's apparatus, yielded remarkable results. For mobile devices, the site scored 99% in Performance and impeccable 100% for Accessibility, Best Practices, and SEO on Google's PageSpeed Insights. For desktops, we achieved flawless 100% across categories. Using Cloudflare's instrument, the site obtained 95% for mobiles and 99% for desktops. These splendid outcomes demonstrate our design choices' effectiveness and refinements, fostering a seamless and responsive user experience.

Employing EJS for page rendering, the website features a main index file with the head of the bar containing meta tags, Bootstrap and Bootstrap icons CSS, and custom CSS. The body presents a noscript tag for users with JavaScript disabled, a main-content div for loading other pages, two necessary modals, and the loading of jQuery, Bootstrap, and custom JavaScript.

Partial files merge into the central content area, forging a continuous one-page design sans redirects.

6.2 Navigation and Layout

The navigation bar adapts to diverse screen dimensions for optimal utility. On larger displays, the navbar resides on the side, expanding upon mouseover, while smaller screens showcase a button revealing the navigation menu when clicked. The navbar, housing the Sus Guru icon and logo, is followed by Home, Scan, Latest Scans, API, Contact, Privacy Policy, and About options. Selecting any item loads corresponding content in the primary body without site departure, ensuring a fluid and seamless browsing adventure.

The dynamic figure 6.2 above vividly captures the layout of the interface, featuring the sleek navigation bar strategically positioned on the right side of the screen. Dominating the center stage is the eye-catching "card" which serves as the primary content showcase, seamlessly loading all the partial files with finesse.

Figure 6.2: Screenshot of Sus Guru's Layout

6.2.1 Desktop Navbar

In the desktop navigation bar illustration 6.3 below, three distinct states are presented, showcasing the dynamic behavior of the nav bar. From left to right:

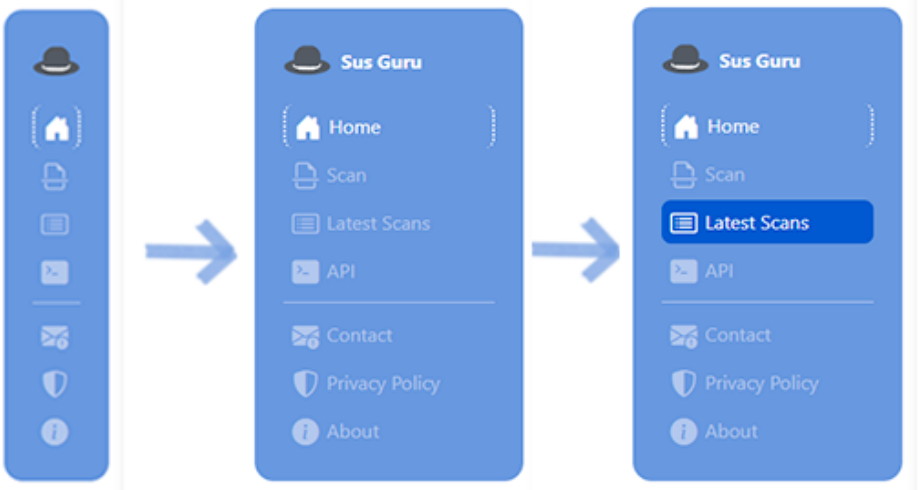

Figure 6.3: Screenshot of Sus Guru's Desktop navigation bar

1. **Closed state:** The navigation bar is in its default, compact form. The icon representing the current page the user is on is highlighted with a border, indicating the active selection.
2. **Expanded state:** As the user hovers their mouse over the nav bar, it expands to reveal additional information, such as labels for each icon. This expanded view provides further context and aids in navigation.

3. **Hover effect:** The third state on the right demonstrates the visual effect when a user hovers over a particular item in the expanded nav bar. This hover effect offers visual feedback to the user, indicating their current selection before clicking on an item.

6.2.2 Mobile Navbar

The following figure 6.4 illustrates the adaptability of the mobile navigation bar, which gracefully transitions between its two states. In the closed state on the left, an unobtrusive floating action button (FAB) is nestled at the top right corner, beckoning users to interact. Upon clicking the FAB, the navigation menu emerges in the center of the screen, with the background gently blurred to emphasize focus.

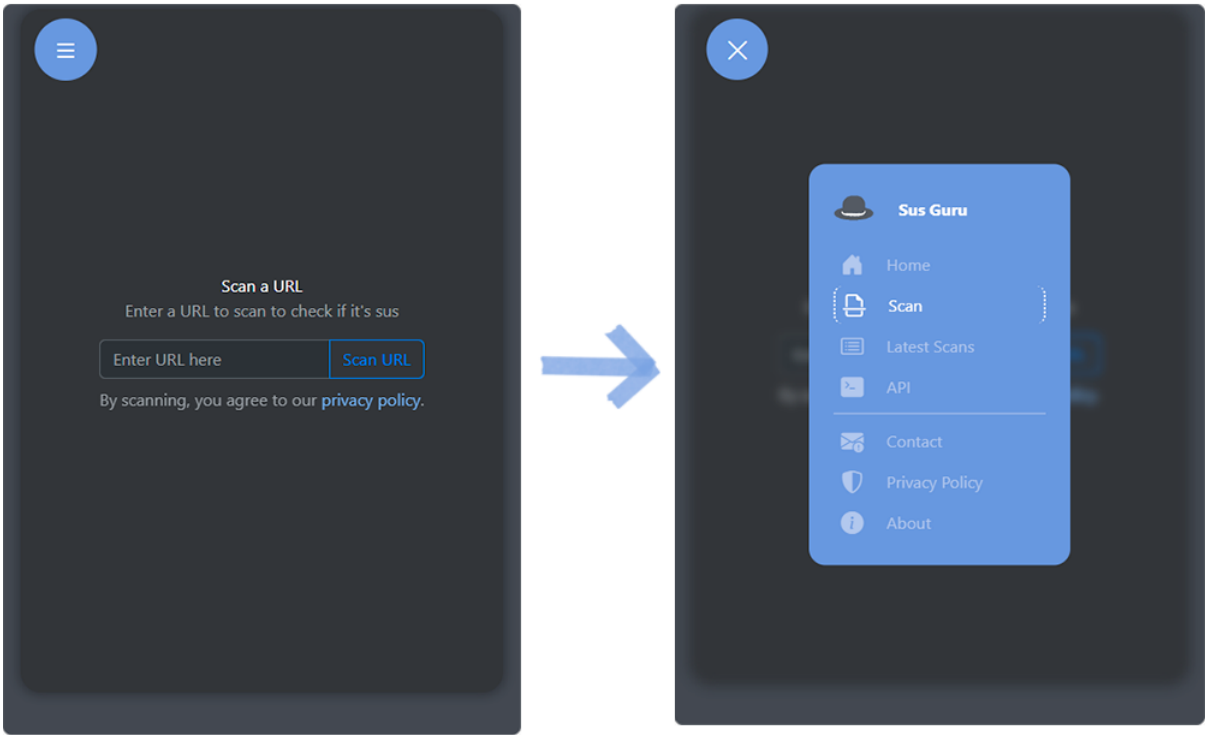

Figure 6.4: Screenshot of Sus Guru's Mobile navigation bar

As showcased in the opened state on the right, selecting a menu item effortlessly transports the user to the desired page. Simultaneously, the navigation menu retracts, and the background blur dissipates, returning the user's attention to the main content.

6.3 Home

The Home segment offers users an encapsulation of Sus Guru's services and objectives, fashioned to be pristine and minimal for a welcoming initial impression. Upon arrival, all messages and

elements are animated utilizing the animate.css library, crafting a visually enticing and captivating gateway to the platform. A call-to-action button, directing users to the Scan page, imparts users with a transparent understanding of Sus Guru's provisions and lures them to investigate further.

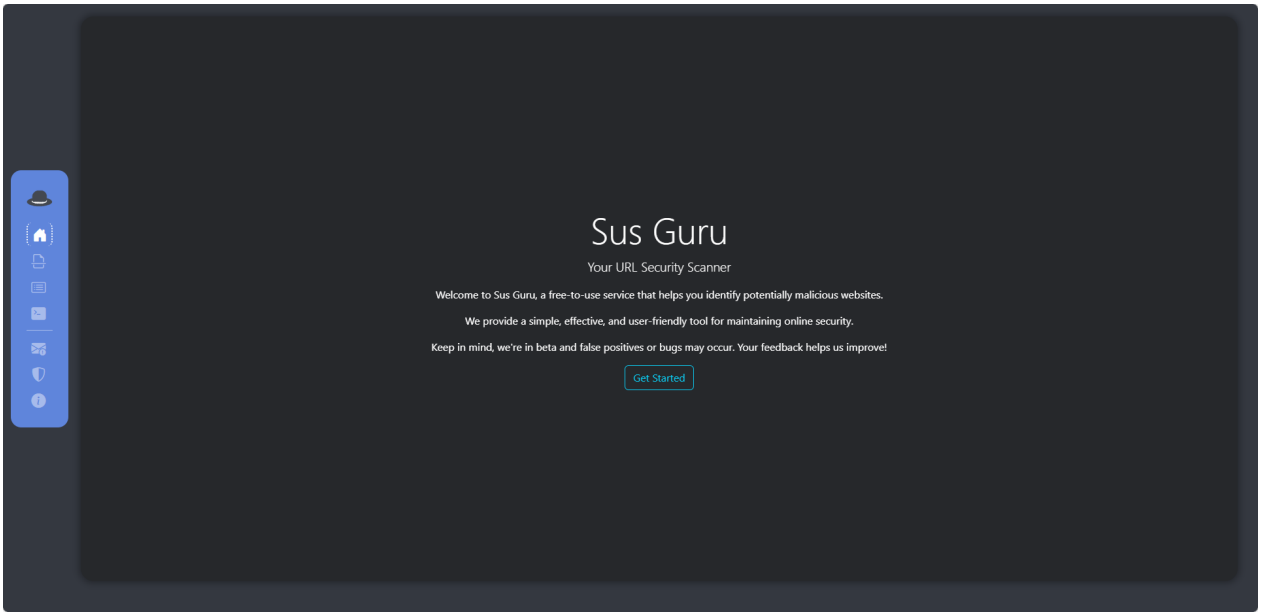

Figure 6.5: Screenshot of Sus Guru's Home Page

By maintaining the Home section uncluttered, minimal, and animated, Sus Guru secures a captivating and user-friendly inauguration to the platform. This tactic enables users to swiftly grasp the platform's intent and capabilities while urging them to explore its features and instruments.

6.4 Scan

The Scan section represents a pivotal aspect of Sus Guru's user interface, permitting users to effortlessly submit a website URL for examination and obtain an extensive report on its security and latent hazards. This section's design ensures an intuitive and engaging experience, keeping users apprised throughout the scrutiny process.

When users select the Investigate item in the navigation bar, a straightforward input field appears, prompting users to enter the desired URL. After submission, the inspection process commences, usually requiring 5 to 20 seconds for completion. Throughout this interval, users maintain server-side event connectivity, facilitating real-time communication and updates.

To sustain user engagement during the waiting period, a Sherlock Holmes-inspired pondering gif materializes on the screen, complemented by impromptu live messages displayed atop it. These messages serve various functions, such as offering valuable advice, revealing intriguing facts, or merely expressing encouragement. The Sherlock gif ideally suits this scenario, humorously portraying Sus Guru's diligent examination of the website in question, making the waiting process

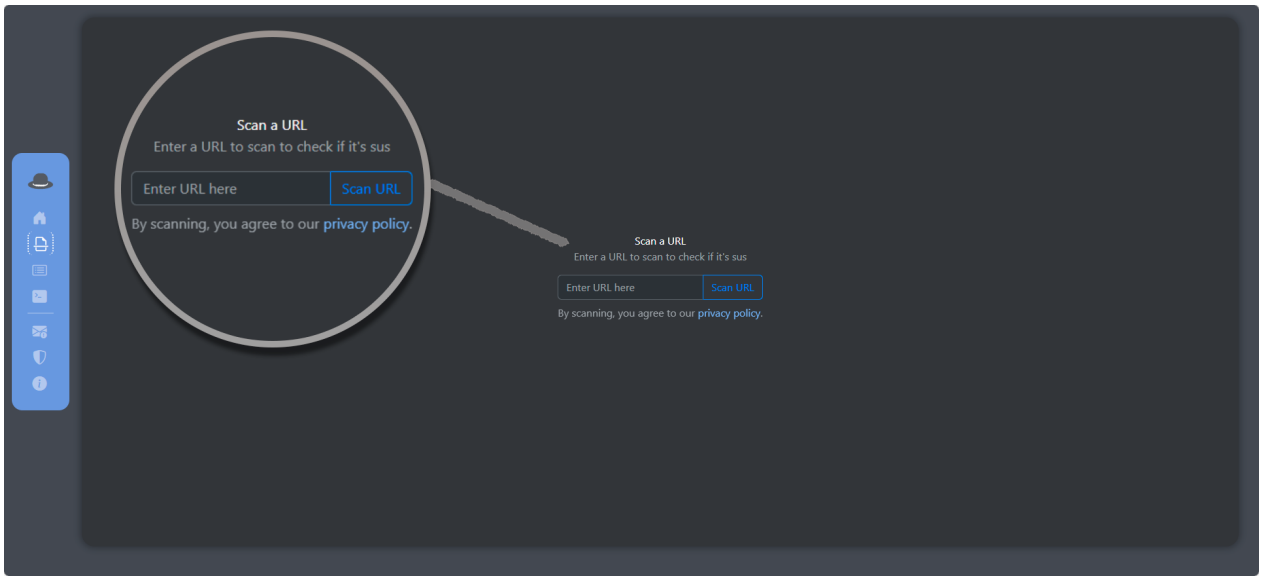

Figure 6.6: Screenshot of Sus Guru's Scan Page

more pleasurable.

As the inspection advances, a progress bar materializes on the screen, indicating the analysis's completion status visually. This progress bar helps manage user expectations and diminishes the perception of waiting duration.

Upon inspection completion, users are automatically directed to the report page to review the analysis results. The report comprises in-depth information about the website's security and any identified potential risks. By seamlessly navigating users to the report page, the user experience remains uninterrupted.

In conclusion, Sus Guru's Investigate section of the user interface aims to deliver an intuitive, engaging, and informative experience. By employing server-side events for real-time updates, integrating progress indicators, and ensuring smooth transitions to the report page, the inspection process constitutes a fundamental part of the platform's overall user experience.

6.5 Latest Scans

The Latest Scans segment showcases a roster of the freshest scans conducted by users, along with corresponding safety evaluations, ID, Timestamp, URL, Prediction, and Votes. Users can select a report ID to access it or click the URL of a secured website to initiate it in a fresh tab. If the link is anticipated malicious, a modal materializes, inquiring if the user is assured about venturing to the malevolent site. The votes represent a combination of downvotes and upvotes

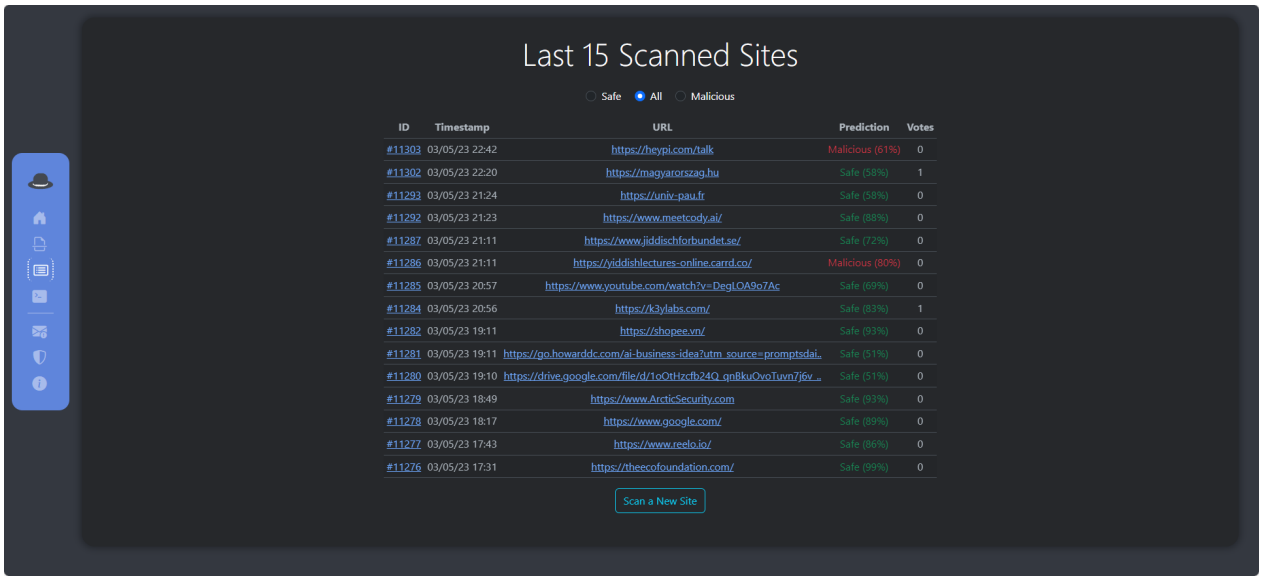

Figure 6.7: Screenshot of Sus Guru's Latest Scans Page

6.6 API

In the beginning, Sus Guru's API routes were documented in a partial HTML file, proffering a rudimentary yet restricted portrayal of the accessible routes. To augment the user experience, we transitioned to harnessing Swagger for the API documentation. Swagger affords a more visually engaging and interactive environment, clarifying users' comprehension and experimentation of the available API routes.

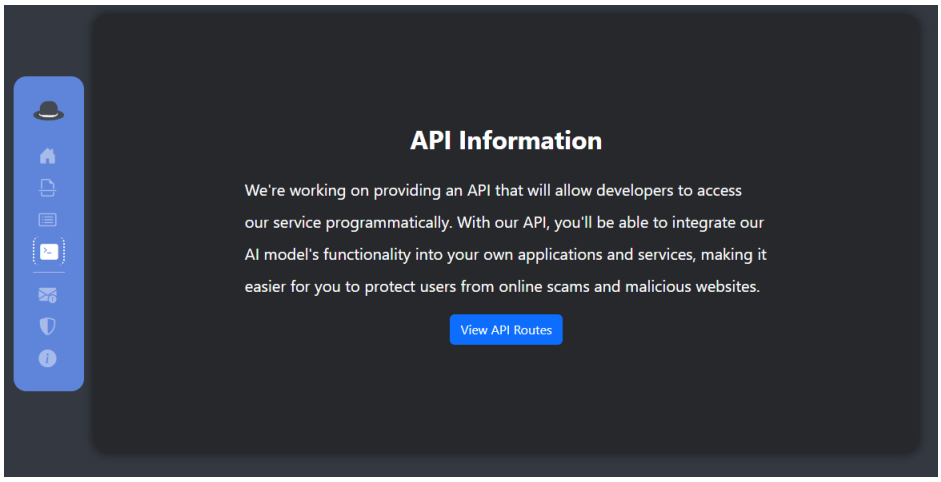

Figure 6.8: Screenshot of Sus Guru's API Page

The Swagger UI shown in figure 6.9 was not incorporated directly into the API page; instead, it was positioned at the /swagger route. This choice was made to avert loading the requisite libraries for Swagger in the primary index file, potentially impeding the initial page load for all users. By situating the Swagger UI on a distinct route, solely users captivated by the API documentation need

to load these libraries, guaranteeing a more expeditious and streamlined experience for the majority of users.

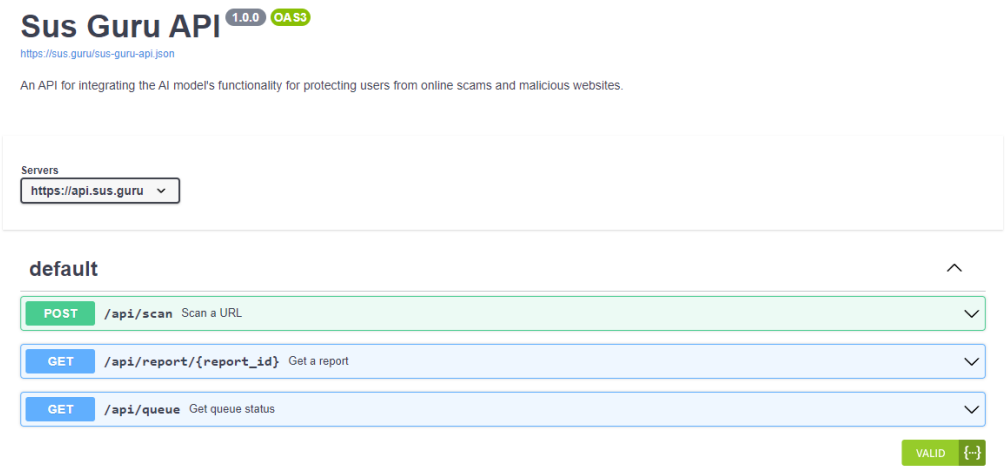

Figure 6.9: Screenshot of Sus Guru's Swagger Page

6.7 Report Page

The Report Page is a crucial component of Sus Guru's user interface, providing users with data about a website's safety. A primary objective of Sus Guru is preserving a user-friendly interface that steers clear of excessive technicality. Consequently, the parameters assembled during the scanning procedure are not overtly manifested on the Report page. Alternatively, the page fixates on showcasing the most pertinent and straightforward information for users.

The conspicuous components on the Report Page consist of:

1. **Screenshot:** A snapshot of the reported website is presented, with any identified nudity obscured and tagged as "Contains nudity" to safeguard users from explicit material.
2. **Report ID:** The distinct identifier allocated to the particular report. (#23411)
3. **URL:** The website URL that underwent scanning.
4. **Timestamp:** The date and time when the scan transpired.
5. **Prediction:** The safety prediction for the website (Safe or Malicious), accompanied by the probability score.
6. **Community Score:** This section encompasses vote buttons, permitting users to voice their opinion on the website's safety.

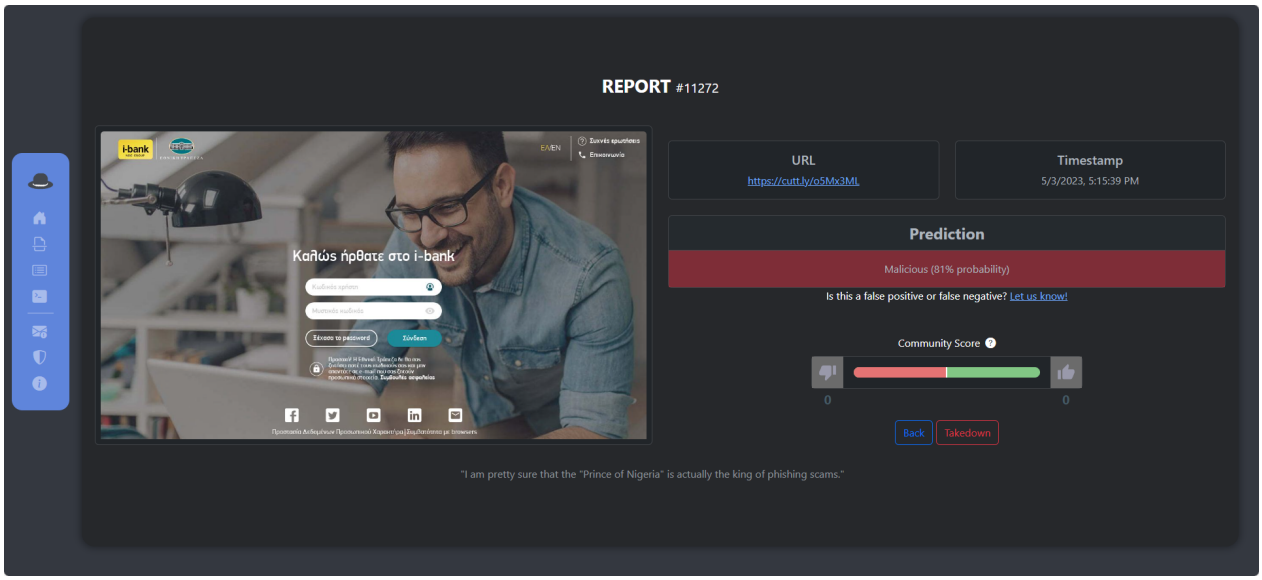

Figure 6.10: Screenshot of a random malicious report

7. **Takedown Button:** This button materializes if the report predicts the website as malicious, offering users choices to act against the site.

Incorporating a hint of humour and individuality into the user interface, an arbitrary comical security-related message is shown at the base of the Report Page. These messages cater not only to amuse users but also to provide subtle security pointers, assisting users in recollecting the information with a giggle. This method contributes to a more affable and captivating user experience, distinguishing Sus Guru from other security instruments.

By centering on presenting solely the most relevant and easily comprehensible data, the Report Page achieves its aim of delivering a user-friendly encounter that fosters online safety and enables users to make well-informed decisions regarding the websites they frequent.

6.8 Community Score

Delving into the heart of the Report Page, one cannot help but notice the Community Score section - an intriguing aspect designed to encourage user interaction. Here, individuals have the power to express their thoughts on a website's safety by casting their votes, either in favor or against.

If a report is deemed malicious, a new button labelled "Takedown" emerges adjacent to the back button, as it's shown in the following figure 6.11

This collaborative approach to assessing security fosters a sense of unity, allowing the community to converge upon a mutual understanding of a site's safety. The amalgamation of diverse opinions contributes to a more comprehensive evaluation.

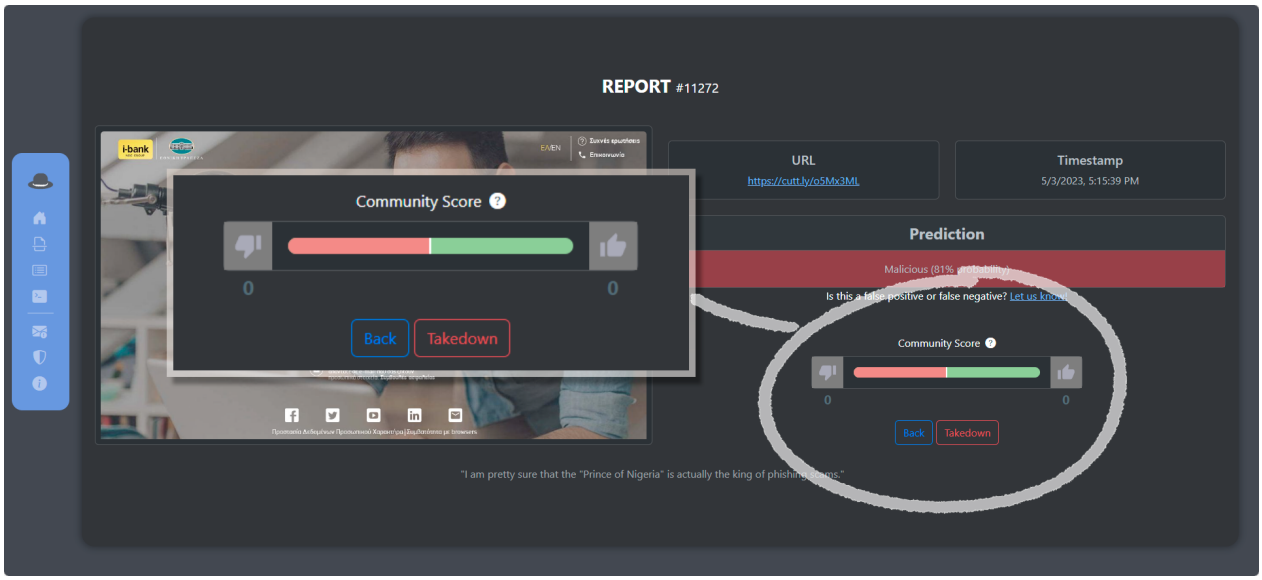

Figure 6.11: Screenshot of a random malicious report showcasing the community score

As users exercise their right to vote, the dynamic progress bar updates. Alongside with the voting buttons, this visual aid succinctly reflects the ongoing battle between upvotes and downvotes. This visual indicator provides users with an at-a-glance understanding of the overall community sentiment regarding the safety of the website in question.

6.9 Takedown Options

Within the labyrinthine depths of the Report Page, Sus Guru has hidden an invaluable weapon - the Takedown button. Emerging only when malicious intent is detected, this button nestles inconspicuously beside the back button, biding its time.

Once summoned by a click, the Takedown button unfurls a modal, revealing a trifecta of formidable options, each designed to combat the sinister threat posed by malicious websites. These alternatives empower users to actively thwart the proliferation of harmful content, thereby fostering a more secure digital landscape.

Enumerated below are the available options:

1. **”Mail website’s registrar”:** This choice proffers a mailto link that connects users to the registrar’s abuse email, granting them the ability to voice their apprehensions about the nefarious site.

2. **”Report to Microsoft Security Intelligence”:** With this selection, users are transported to Microsoft’s Security Intelligence reporting tool, enabling them to report the nefarious site for further investigation and possible blacklisting of the malevolent site.

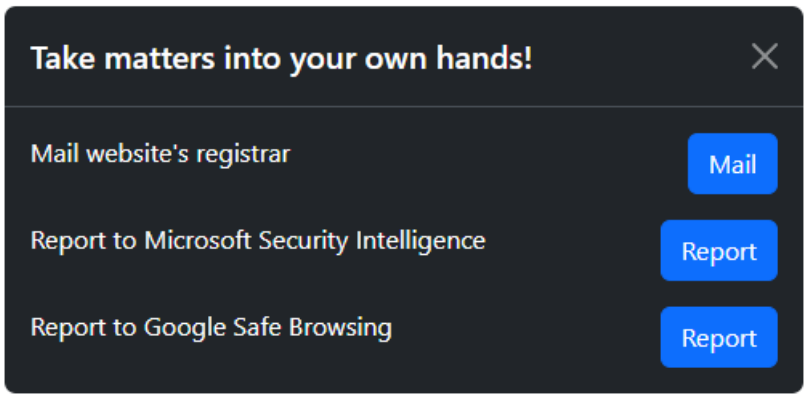

Figure 6.12: Screenshot of a takedown modal

3. **"Report to Google Safe Browsing":** In a similar vein, this option ushers users toward Google's Safe Browsing reporting tool, facilitating the submission of the malicious site for examination and potential banishment to Google's digital underworld.

Sus Guru encourages a safer online environment and inhibits the dissemination of harmful content by giving users the ability to take action against malicious websites through these Takedown options.

In conclusion, the user interface of Sus Guru's Report Page has been carefully created to give users useful information about the security of a website, as well as to encourage community engagement through the Community Score feature and give users the opportunity to take action against malicious websites through the Takedown options. With this thoughtful design, users are guaranteed a thorough and user-friendly experience that encourages online safety and gives them the power to choose the websites they visit wisely.

6.10 Admin Panel

The Admin Panel is a dedicated section of Sus Guru's user interface designed specifically for, well... administrators. This panel offers essential tools and features that allow admins to monitor the platform's performance, manage reports, and identify possible false positives or negatives. As Sus Guru continues to evolve, the Admin Panel will be further developed to include additional controls and functionality.

The Admin Panel's main features are as follows:

1. **Statistics:** For the previous three days, the Admin Panel has provides up-to-date data on the total, safe, and malicious reports. Administrators may use this data to monitor platform performance, identify patterns, and make well-informed choices regarding system upgrades and changes.

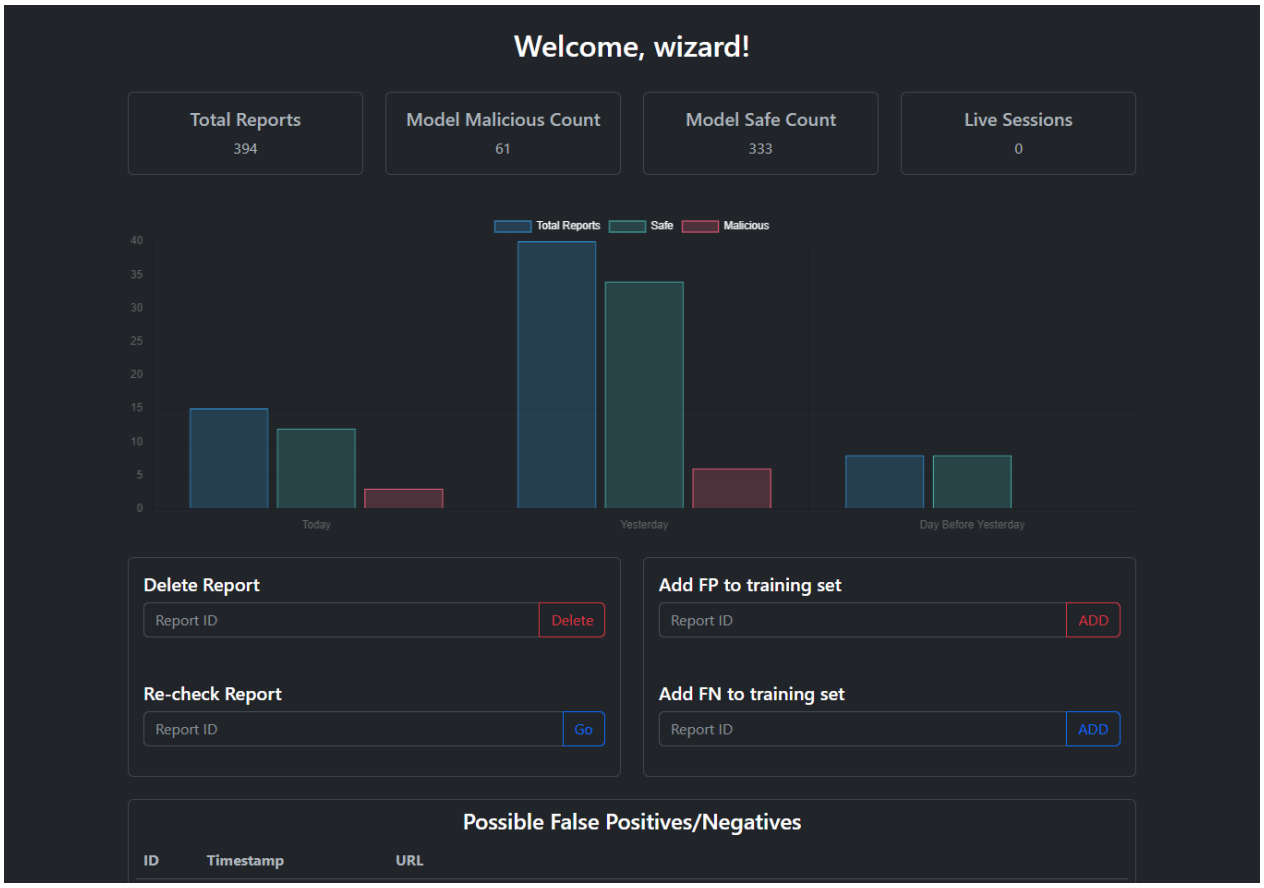

Figure 6.13: Screenshot of Sus Guru's Admin Panel

2. **Report Management:** Admins have access to controls that let them remove and double-check certain report IDs. With the help of this functionality, platform administrators may protect the data's integrity and address user questions or concerns about certain reports.

3. **Possible False Positives/Negatives Table:** This table displays reports that may contain false positives or negatives, such as cases where a website has been predicted as malicious but has a majority of user votes indicating it is safe, or vice versa. By identifying these potential discrepancies, admins can review the accuracy of the platform's predictions and take corrective action as needed.

The Admin Panel will be improved to include new controls and features as the platform grows and evolves, ensuring that administrators have the tools they need to maintain and improve Sus Guru's performance and user experience.

Finally, the Admin Panel is an important part of Sus Guru's user interface, providing administrators with a variety of tools and functions to assist them administer the platform, analyse its performance, and discover areas for development. The Admin Panel plays a significant part in Sus Guru's continuous success and growth by offering these critical insights and capabilities.

Possible False Positives/Negatives

ID	Timestamp	URL	Prediction	Votes
#11002	2023-04-27T19:56:30.561344+03:00	https://telekom.de	Safe (83%)	-1
#11150	2023-04-29T14:23:41.561115+03:00	https://docs.google.com/forms/d/e/...	Malicious (85%)	1
#11152	2023-04-29T14:46:55.664019+03:00	https://nexus.bulletnode.com	Malicious (68%)	1
#11226	2023-05-01T01:40:35.431751+03:00	https://blackhatresource.com/	Malicious (54%)	1
#11010	2023-04-27T20:54:25.666228+03:00	https://spirosconsulting.com/	Malicious (69%)	1
#11023	2023-04-27T23:16:45.890631+03:00	https://mail.bulutepostam.com	Malicious (73%)	1
#11099	2023-04-28T15:17:23.581225+03:00	https://godmode.space/	Malicious (76%)	1
#11103	2023-04-	https://ytinfo.com/	Safe (74%)	-1

Figure 6.14: Screenshot of Sus Guru's Admin Panel's possible False Positive/Negative table

6.11 Contact, Privacy Policy, and About

The remaining portions of the user interface (Contact, Privacy Policy, and About) provide more information on Sus Guru's services, contact information, privacy practices, and the project's information. These sections are intended to be informative and simple to navigate, allowing users to get the information they need quickly and efficiently.

Notably, the Contact form is configured to send a webhook to both admins on Discord when a new message is submitted, complete with tastefully designed embeds. The immediate notification system enables the team to quickly review and respond to user inquiries, ensuring effective communication and support. Sus Guru also has its own support email (support@sus.guru), which is routed through Cloudflare and configured with Google's servers to efficiently send and receive emails.

With the assistance of a skilled lawyer, the Privacy Policy page provides critical information to cover common issues, threats, and abuse. This policy guarantees that users are aware of Sus Guru's activities and that the platform complies with legal and privacy standards.

The About section provides a brief overview of the project, including its objectives, inspiration, and basic information about the team behind Sus Guru. This section helps users gain a better understanding of the platform's purpose and the people responsible for its development.

In conclusion, Sus Guru's user interface design effectively accomplishes its objectives of being

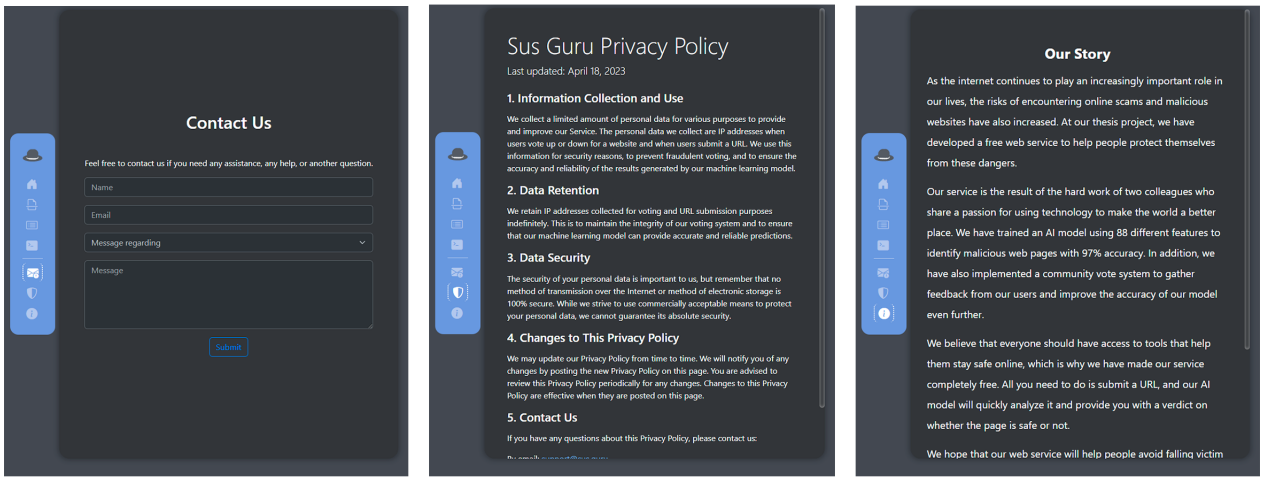

Figure 6.15: Screenshot of a Contact Form, Privacy Policy, and About Page

user-friendly, browser-compatible, and adaptable to a variety of devices. By thoughtfully considering layout, navigation, and functionality, the user interface offers an engaging and seamless experience for users while ensuring they can access and interact with the platform's features proficiently. The integration of efficient communication tools, like Discord webhooks and a dedicated support email, further enhances the user experience and enables the Sus Guru team to offer prompt and effective support.

Chapter 7

Discussion

As we embark on discussing the comparisons between our malicious website detection system and other prominent systems, it's important to recognize that our tool is just another approach in the ongoing battle against phishing and malicious websites. The fight against cyber threats demands a multifaceted and collaborative effort, leveraging every available resource and methodology. By examining the features, strengths, and limitations of various methods, we can better appreciate the unique advantages of our approach and its place within the broader landscape of malicious website detection tools.

7.1 Insights

While developing our parameters, we faced several obstacles and unexpected findings. One major challenge was testing new parameters on phishing or malicious sites, as cybercriminals often used various techniques to avoid detection.

7.1.1 Avoidance Methods

A common strategy involved using HTTP failure codes, which created the false impression that a phishing link was no longer active. However, when retesting later, the site would often reappear. This tactic aimed to remove the site from blacklists or trick crawlers into thinking the site was down. Some sites even enhanced this method by using a mix of factors, such as visit frequency, user agents, and other variables to predict whether it was a new user, a returning user, or a blacklist crawler checking the site. Unfortunately, we could not gather solid evidence to support this claim, as detecting such a site often led to us being blocked.

Another technique, mainly observed in many openresty systems, used a base64-encoded script running within the main content of the HTML document through window.park. The script would send browser and IP statistics to the criminals' API, which would then either allow the phishing

Figure 7.1: Screenshot of a random malicious malicious website's code

site to load or display a "regular" website without malicious content. This was another method designed to deceive crawlers and bots. The figure 7.1 above this section shows the two states of this technique, with the left side illustrating the script that granted approval, and the right side revealing the "ordinary" website.

7.1.2 Server Types

In this section, we'll analyze the top 25 server types for each category, highlighting any notable distinctions between normal and malicious websites.

Prominent server types with higher presence in malicious websites compared to normal websites:

1. **Openresty:** 8.91% in malicious websites vs. 1.25% in normal websites, signifying a much higher presence in malicious websites.
2. **LiteSpeed:** 4.42% in malicious websites vs. 1.07% in normal websites, indicating a higher presence in malicious websites.
3. **GSE:** 2.07% in malicious websites vs. not in the top 25 for normal websites, suggesting a higher presence in malicious websites.

Significant server types with lower presence in malicious websites compared to normal websites:

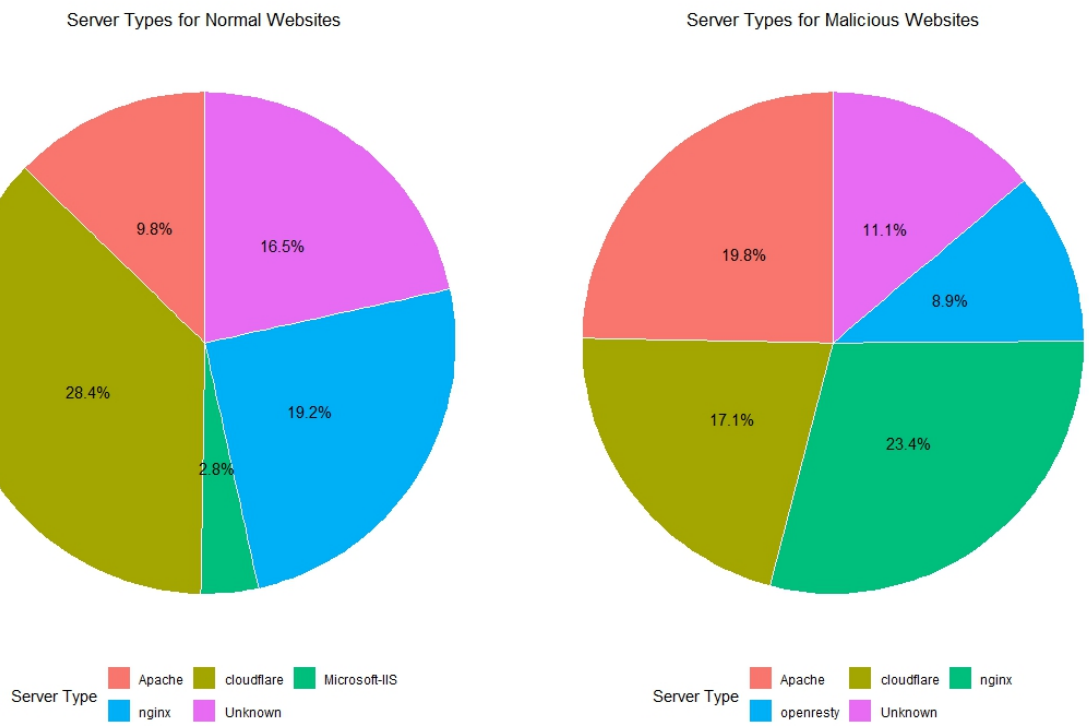

Figure 7.2: Chart of the top 5 server types for Malicious and Normal websites

1. **Cloudflare:** 17.1% in malicious websites vs. 28.4% in normal websites, demonstrating a lower presence in malicious websites.
2. **Microsoft-IIS:** 0.544% in malicious websites vs. 2.83% in normal websites, pointing to a lower presence in malicious websites.
3. **ESF:** 0.661% in malicious websites vs. 1.49% in normal websites, displaying a lower presence in malicious websites.

Other interesting observations:

1. AmazonS3, Netlify, Google Frontend, Vercel, and GitHub.com appear in both normal and malicious website lists but exhibit no significant difference in distribution.
2. Some server types, such as DPS, PHP, and HTTP server (unknown), are only present in the top 25 list for malicious websites, potentially warranting further exploration.

This analysis of the top 25 server types exposes that certain server types have a substantially higher or lower presence in malicious websites compared to normal websites. Understanding these differences could help grasp the infrastructure preferences of malicious actors and serve as a foundation for additional investigation.

7.2 Thoughts

In this study, we investigated and analysed the findings of multiple machine learning models for detecting malicious websites. Our findings have important implications for malicious website detection and shed light on the advantages and disadvantages of various algorithms. We have highlighted the distinctive features and benefits of our technique by comparing it to other similar systems in the literature.

Since the launch of our malicious website detection service, we've seen an interesting mix of traffic from indexers, crawlers, and bots. Indexers and crawlers are automated programs that explore the web in order to collect information for search engines or other reasons, whereas bots are software applications that conduct automated activities. The sheer volume of automated traffic was notable, stressing the importance of robust detection systems capable of handling such a diverse range of user agents.

In addition, we observed that users were fast to test the security of our service by looking for SQL injections and XSS attacks. Fortunately, their attempts were futile, demonstrating the robustness of our system. This conduct emphasizes the significance of constantly upgrading and strengthening our security systems in order to remain ahead of potential attackers.

As the website gained traction, it was placed on Y Combinator's Hacker News, attracting even more users to test the system and scan websites via both the web interface and API. This puplicity resulted in useful feedback from users, including website owners who had false positives or just wanted to know how it worked. Such interactions help to improve the overall performance of the service by refining our detection models.

After we set up our public API routes on RapidAPI, we gained more users who found value in our service. This aided in broadening our reach and encouraging additional community participation.

Our primary goal for future advances is to go fully open-source. There are advantages to open-source development, like enhanced collaboration, openness, and progressive refinement; however, potential risks must also be acknowledged. Malicious actors could scrutinize the source code to bypass safeguards or take advantage of system weaknesses.

Still, we maintain that the pros of open-source development surpass the cons. By engaging a diverse community of developers and security experts, we can persistently advance the system and adapt to the ever-shifting landscape of dangerous websites and cyber hazards.

In summary, our malicious website detection service has yielded promising outcomes and captured the attention of a broad array of individuals. As we transition to an open-source framework, we anticipate that the merits of collaboration, transparency, and ongoing improvement will contribute to a more effective and resilient service, ultimately fostering a safer online space for all.

Chapter 8

Limitations, Future Work, and Conclusion

8.1 Limitations

While our project is ambitious and innovative, it has encountered several limitations that have influenced the scope and implementation of the system. The primary constraints were time and budget. We began the project relatively late, with only 3-4 months to complete the entire process, including research, reviewing related papers, developing the backend architecture and frontend, and writing the paper. This tight schedule prevented us from fully exploring all possible avenues and implementing certain features.

Budgetary constraints further limited our access to potentially beneficial external APIs and resources, as we aimed to maintain the system's free availability. We were cautious about relying on external APIs due to the risk of sudden changes in their licensing or availability. Our commitment to keeping our system free forever meant that we had to avoid using paid or pay-as-you-go libraries.

8.2 Future Work

Despite limitations, ample room exists for future research, improvements, and extensions to our work. Potential areas for future development include:

1. **CERT team auto-mailing system:** An automated system notifying Computer Emergency Response Teams (CERTs) when a phishing website is detected allows rapid response and takedown, reducing scammers' window of opportunity and minimizing phishing campaign impact.
2. **Features and enhancements:** Improving the user interface and model makes the system more accessible and user-friendly while increasing the phishing detection algorithm's accuracy and performance. This might involve adding new features, refining existing ones, or addressing research-identified shortcomings.

3. **Model improvement:** Enhancing the performance of the model by identifying and eliminating less important features, reducing the number of trees required, and using ensemble methods, such as stacking or boosting, to combine the strengths of multiple models, resulting in a more efficient and accurate malicious website detection system.

4. **DNS blacklists:** Creating a system to produce and manage DNS blacklists with malicious domains identified by the model, these lists can be integrated into firewalls and filtering systems for automatic blocking. This improves overall cybersecurity and lowers the likelihood of encountering malicious websites.

5. **Discord bot development:** A Discord bot using the public API filters malicious messages in real-time, protecting server owners and users from phishing threats. This application could demonstrate the system's incorporation into other messaging platforms.

6. **Free Chrome extension for website analysis:** A user-friendly Chrome extension lets users analyze websites with one click. Integrating the system's API into a browser extension helps users assess website legitimacy before interaction, reducing phishing scam vulnerability.

7. **Automatic email system for phishing email detection:** A system using honeypot accounts to identify phishing emails offers valuable data on new phishing campaigns. The system could automatically analyze incoming emails for phishing links and refine the detection algorithm.

8. **Malicious website auto-scraper utilizing search engine APIs:** By leveraging search engine APIs, an automated web scraper could be developed to continuously discover and identify new phishing sites. This would allow the model to be updated with the latest threats and improve its ability to detect and prevent attacks from newly created malicious websites.

9. **Parameter-based scoring algorithm:** A scoring algorithm weighing various system-collected parameters enables a more nuanced and accurate phishing risk assessment. This contributes to overall prediction accuracy and enhances the system's ability to detect and prevent phishing attacks.

8.3 Conclusion

Embarking on this project, our aim was to create a free, accessible system for a safer web experience for everyone. Using technologies like Node.js, PostgreSQL, and assorted machine learning algorithms, we crafted a sturdy and dependable tool to detect phishing websites.

Fascinatingly, a casual conversation between the authors sparked the project idea. One struggled to find an intriguing thesis topic, while the other was amusingly showing an automated script generating fake personal and credit card information, that would then go through a four-step phishing process, bombarding the phishing website with fake data. In the midst of our lighthearted chat, it hit us - the perfect research idea!

Our solution works as a standalone system and has potential for integration into existing systems, boosting cybersecurity capabilities. AI's role in this context is nothing short of enchanting, as it actively contributes to the ongoing battle against phishing and malicious intent, which only worsens daily.

Despite challenges, our system showcases potential to considerably contribute to the cybersecurity field. We believe our work, combined with future enhancements and extensions, can create a more secure online environment for all users.

Bibliography

[1] S. Das and T. Nayak, "Impact of cybercrime: Issues and challenges," *International journal of engineering sciences & Emerging technologies*, vol. 6, no. 2, pp. 142–153, 2013.

[2] K. Jaishankar, "Establishing a theory of cyber crimes," *International Journal of Cyber Criminology*, vol. 1, no. 2, pp. 7–9, 2007.

[3] C. Stupp, "Fraudsters used ai to mimic ceo's voice in unusual cybercrime case," Aug 2019.

[4] "How i broke into a bank account with an ai-generated voice," Feb 2023.

[5] "Internet Crime Complaint Center(IC3) — Home Page — ic3.gov." urlhttps://www.ic3.gov/, 2023. [Accessed 06-May-2023].

[6] Python Software Foundation, "About python™ — python.org," Accessed: April 23, 2023.

[7] Node.js Foundation, "About — node.js," Accessed: April 22, 2023.

[8] Bootstrap, "Bootstrap · the most popular html, css, and js library in the world.," Accessed: April 22, 2023.

[9] jQuery Foundation, "jquery," Accessed: April 22, 2023.

[10] EmbeddedJS, "Ejs – embedded javascript templates," Accessed: April 22, 2023.

[11] M. Sameen, K. Han, and S. O. Hwang, "Phishhaven - an efficient real-time ai phishing urls detection system," *IEEE Access*, vol. 8, pp. 83425–83443, 2020.

[12] H. Faris and S. Yazid, "Phishing web page detection methods: Url and html features detection," *IoTaIS 2020 - Proceedings: 2020 IEEE International Conference on Internet of Things and Intelligence Systems*, pp. 167–171, 1 2021.

[13] X. He, L. Xu, and C. Cha, "Malicious javascript code detection based on hybrid analysis," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 2018-December, pp. 365–374, 7 2018.

[14] F. Yahva, R. I. W. Mahibol, C. K. Ying, M. B. Anai, S. A. Frankie, E. L. N. Wei, and R. G. Utomo, "Detection of phising websites using machine learning approaches," *2021 International Conference on Data Science and Its Applications, ICoDSA 2021*, pp. 40–47, 2021.

[15] Y. Cao, Y. Gao, R. Tan, Q. Han, and Z. Liu, "Understanding internet ddos mitigation from academic and industrial perspectives," *IEEE Access*, vol. 6, pp. 66641–66648, 2018.

[16] P. Patil, R. Rane, and M. Bhalekar, "Detecting spam and phishing mails using svm and obfuscation url detection algorithm," *Proceedings of the International Conference on Inventive Systems and Control, ICISC 2017*, 10 2017.

[17] J. . Setyanto, A. . Alarfaj, F. K. . Alreshoodi, J. Moedjahedy, A. Setyanto, F. K. Alarfaj, and M. Alreshoodi, "Ccrfs: Combine correlation features selection for detecting phishing websites using machine learning," *Future Internet 2022, Vol. 14, Page 229*, vol. 14, p. 229, 7 2022.

[18] K. S. Swarnalatha, K. C. Ramchandra, K. Ansari, L. Ojha, and S. S. Sharma, "Real-time threat intelligence-block phising attacks," *CSITSS 2021 - 2021 5th International Conference on Computational Systems and Information Technology for Sustainable Solutions, Proceedings*, 2021.

[19] G. Sonowal and K. S. Kuppusamy, "Phidma – a phishing detection model with multi-filter approach," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, pp. 99–112, 1 2020.

[20] Z. Fa, G. G. Geng, Z. W. Yan, and X. D. Lee, "A robust internet abuse detection method," *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, vol. 2018-January, pp. 1712–1715, 7 2017.