THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

MALICIOUS WEBSITE REDIRECTION DETECTION

by

Wardsy

School of Information Technology and Electrical Engineering,
The University of Queensland.

Submitted for the degree of
Bachelor of Engineering
in the field of Software Engineering

NOVEMBER 2021.

Acknowledgments

I would like to acknowledge my supervisor Dr Slivia Ma for her guidance throughout this project. In particular, her overview of the data collection process and assistance with appropriate classification models. Furthermore, Dr Ma would regularly meet with me, and provide feedback based on fortnightly updates. I would also like to acknowledge fellow student Mugdha Khatavkar for her work on building the initial web crawler script. This script was used to collect an initial group of URLs from a search engine that were later labelled and included in my dataset.

Abstract

Website redirection is a technique used to make web pages available under many URL addresses. Traditionally, the technique is most commonly implemented through buttons, links and embedded media in a website. Despite the many beneficial use cases, website redirections are a common way for users to be led to malicious websites. Previous literature into preventing malicious website redirections is outdated, and many attackers now use tools to evade detection from current systems.

This study proposes a method that classifies malicious website redirections using text and image content of a webpage, thus denying attackers the opportunity to use evasive techniques. A list of malicious keywords were generated and used as input into a web-crawler that recorded webpage text and image content. This data was then collated and given a binary label as either malicious or benign. Multiple text and image models were built using this dataset, where it was found that when combined, both models predict malicious redirections with over 99% accuracy. These models can act as a basis for further research, as well as a key part of an initial prototype that detects malicious redirections in real-time from a browser.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Website redirection is a technique used to make web pages available under multiple URL addresses [1]. Despite the many legitimate use cases for website redirections, malicious actors often utilise the technique to draw users to harmful websites. Current detection and prevention methods work with relatively high accuracy, however malicious actors are becoming more aware of evasive techniques to overcome these systems. Approximately 45% of internet users have encountered malicious websites through unwanted redirections that end in a loss of data or breach of privacy [2]. By investigating the flaws in existing detection systems, improvements can be made to strengthen resistance against modern evasion techniques. Consequently, the number of data breaches caused by malicious website redirections can be reduced, as well as the financial damages caused by these attacks.

This research proposes a new method for detecting malicious redirections through the use of webpage text and image content. It first uses a series of potentially malicious keywords to generate a list of URLs. Each individual URL is then visited by a web-crawler, where all embedded redirections are triggered. The destination source code and a screenshot for each webpage is saved into a single unlabelled dataset. Based on the nature of the redirection, each data entry is then labelled as either malicious or benign. The text and image content extracted is then preprocessed to form a labelled dataset that is suitable as input for text and image-based machine learning models. This data is split into two partitions, one used to train multiple models and one to validate performance. Using the best performing text classifier as well as an optimally tuned deep learning model on image content, a hybrid model is utilised to maximise the true positive rates on unknown data. This final combination of models can then be used as a foundation for a browser extension that detects malicious redirections in real-time.

Using the above method provides a strong foundation for malicious website redirection classification. It was found that by using an SVM model on text features, the nature of a website redirection could be classified with 98.8% accuracy. Furthermore, when a deep learning model was used on webpage screenshots, malicious redirections were detected with 98.9% accuracy. A hybrid model that utilised both classifiers labelled redirections with 99.14% accuracy in near real-time.

The final implementation of both classifiers was designed in such a way that future iterations can easily add user-browser interactions to the pipeline. This allows the potential for real-world use as well as an effective way to automate and streamline future data labelling. It is the intention that this pipeline can be used in future research to strengthen current methods as well as to form the basis of a consumer product that detects malicious redirections.

1.1 Aims

The aim of this project is to protect legitimate websites and users against redirection attacks. The detection will leverage text and image contents of a webpage to identify malicious website redirections in real-time. The final product will demonstrate how these models can be combined to generate a usable redirection detection tool.

1.2 Scope

1.2.1 In Scope

The following items will be produced as a part of this thesis project:

- A dataset of labelled web pages consisting of key text and image components.
- Binary classification models utilising both text and image features of the generated dataset.
- Analysis of model performance
- A working prototype that demonstrates future implementations using generated models.
- Recommendations on how redirection detection systems can be improved based on the results of the project.

Field	**Description**
from_url	The initial webpage visited using a search API
to_url	The redirected website visited via an *href* element in *from_url*
label	Binary label of *malicious* or *benign*
text_content	The key text elements extracted from the *to_url* webpage
image_index	The index of the screenshot file in a seperate images folder

Table 1.1: Dataset features and descriptions

1.2.2 Dataset

A dataset was created for the purpose of training and testing text and image classification models. The dataset contains approximately 11,000 web pages derived from a variety of initial search terms (see **Table 4.1**). Furthermore, the methodology of data collection means that the dataset consists of high-quality web pages that are frequently visited via search engine results.

The dataset consists of multiple fields that were then used to train the binary classification models. Each row of data was assigned a binary label of malicious or benign, which was determined through a manually labelling process. The fields for each webpage in the dataset are summarised in **Table 1.1**.

1.2.3 Attack Types

In order to collect a relevant dataset, only one specific redirection attack type was replicated. The specific attack that was focused on for data collection was an open redirect attack. This is one of the most common attack types, and is the easiest to implement on an unsecured website [3]. The attack works by altering parameter values in an HTTP GET request, which will redirect a user to any site without validation [4]. Given this attack method is fairly common, collecting a large enough dataset was achievable. Furthermore, since the contents of a webpage is being investigated, an attack should be correctly classified regardless of the method used. Thus, the chosen attack method has been used to simplify the dataset requirements, and has minimal impact on the accuracy of the final models.

1.3 Relevance

This research is highly relevant to the current cybersecurity ecosystem, with every 1 in 13 website requests leading to potentially malicious addresses [5]. By investigating the shared features amongst malicious websites, a solution can be developed that uses these similarities to accurately predict when an attack has occurred. Furthermore, an analysis of the suitability of various classification models will provide deep insights for future research regarding how datasets and model parameters influence results. Additionally, given that the final solution is developed in a way that means people can use the models on any sites, the flaws in the models and dataset can be further studied and improved upon in future iterations.

1.4 Results

Using the generated dataset, individual models were developed using text and image contents and predicted malicious website redirections with high accuracy. The final implementation demonstrates how text and image website content can produce real-time results using certain machine learning algorithms. Furthermore, the various differences in web pages highlight how text and image contents can be used in conjunction to produce more accurate classification results.

Particular flaws with the dataset collection process also provide deep insight into the work required for future research to produce results capable of being used by everyday internet users. The work produced for this thesis provides a strong basis for further work in the field, and demonstrates that this method is a viable solution to address current issues.

Chapter 2

Background

2.1 Website Redirection Attacks

Malicious actors distribute malware throughout the internet via drive-by download attacks [6]. When a user accesses a domain via a direct link or search engine result, the landing page acts as a starting point for a redirection attack. From here, a client is then redirected to an exploit URL via many other redirection domains [6]. Once at the destination site, a user is vulnerable to malware via exploited browser vulnerabilities or direct links disguised as legitimate downloads. After being infected, a user is open to data leaks, loss of control over computer functionality and can be used as a node for distributed denial of service (DDoS) attacks [7].

Malicious websites are typically compromised benign websites built on outdated content management systems. Attackers will use botnets to scan the web for web pages running vulnerable CMS systems, where they will then inject their own malicious code to force redirections when a user visits the original page [8]. An example of this is the KashmirBlack botnet that was first detected by researchers at Imperva, and had successfully infected hundreds of thousand of websites by attacking their content management system platforms [9]. Additionally, sites that have links to potentially harmful downloads also fall into the malicious category. For example, whilst torrent websites themselves are typically benign, the user generated content can contain malicious downloads that infect a computer. Currently, torrent websites infect 12 million users a month with malware [10]. Thus, whilst some redirections occur due to manipulated web pages, others may be caused from website owners intentionally disguising malicious domains within buttons and hyperlinks.

2.1.1 Open Redirect Attacks

Open redirection attacks are the most common type implemented by malicious actors [11]. They arise when applications incorporate controllable data into the target field of a redirection in an insecure way. For example, when a user logs into a site and is authenticated, this successful action can be relayed into the URL, routing the user to the main page of a web application. This attack typically arises from vulnerabilities in the content management system of a website. Once a website has been infiltrated, an attacker can simply modify a URL input to a malicious site of their choosing. An example of how a redirection can lead to a malicious website through an open redirect attack is outlined below.

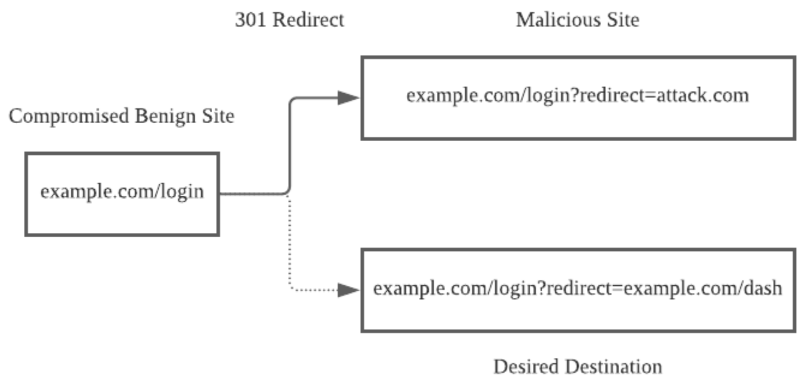


Figure 2.1: Open redirect attack overview

The above figure highlights what an altered parameter value in an HTTP GET request looks like. In this example, the redirect parameter can be modified to a malicious site if a website does not validate the value. A key source of this vulnerability comes from the inconsistency between how browsers decode non-ASCII characters in URLs [12]. For example, if a browser were to decode a non-ASCII character into a question mark from the URL `https://malicious.com%252f.benign.com`, the redirection link would be `https://malicious.com?.benign.com`. Thus, `benign.com` is no longer the hostname, but instead part of the link query. As a result, the browser would instead navigate to `malicious.com`, which is a domain owned by a malicious actor.

Additionally, a majority of browsers and validators will treat a backward slash as a path indicator. If neither the validator nor browser implement this, inconsistencies can lead to to a malicious redirection. For example, if `https://benign.com\`

`malicious.com` is interpreted as `https://benign.com/malicious.com`, then the malicious domain address would be treated as the base domain name and not the sub-domain name. This would thus lead to a redirection to the malicious URL, as opposed to the intended benign site.

Additional open redirect attack methods exist and are exploited regularly by malicious actors. Session restoration URL redirection occurs when a web application preserves the last URL viewed by a user prior to a browser session being terminated [13]. In practice, a user is redirected back the their previous location after being authenticated by the web application. In doing so, a malicious actor is able to modify the return URL to a domain of their choice. This attack stems from existing vulnerabilities in older content management systems that allows a malicious actor to modify the 'returnURL' parameter within an Agent Configuration Object or local configuration file [14].

DOM-based open redirection attacks occur when a script writes controllable data into a sink that can trigger cross-domain navigation [15]. A JavaScript sink is a function or method that can be used for an attack if called. Given that JavaScript can take data directly from a browser, this information can then be passed to a sink that supports dynamic code execution. In order for an attacker to propagate data, the `window.location` object is often used, where a link can be constructed to send a user to a malicious URL. This link is then passed as a payload in the query string of a redirection, thus leading to a potentially malicious redirection if the link is not validated appropriately. This technique is often automated by botnets through scanning the web for vulnerabilities within JavaScript code [15].

2.1.2 IFrame Redirection Attacks

The `iframe` HTML tag is used to nest additional pages within a website [16]. Typically, it is used for slow third-party elements of a webpage to ensure that the extended loading times are only in isolation. The function is appropriate for developers as it means that isolated changes to the `iframe` element do not directly effect the other page elements. Whilst useful, the function is also prone to exposure by malicious actors. Attackers can embed malicious web pages within an `iframe` tag element. In doing so, a user would access a benign URL but would instead be shown a malicious site with potentially harmful code. This technique is particularly common for phishing attacks, as the trust of a benign URL can be exploited by an attacker when embedding malicious forms and buttons within a webpage. An example of how an `iframe` redirection occurs is outlined in the figure below.

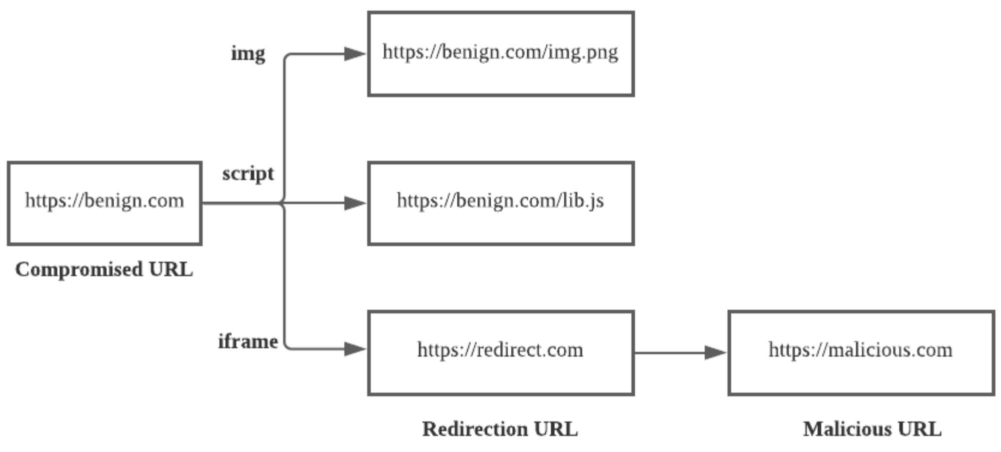


Figure 2.2: IFrame redirection attack overview

Figure 2.2 shows how the `iframe` tag can be used to redirect a user to a malicious URL. Furthermore, it shows other key webpage elements to describe what the user may see. All elements of the webpage will appear normal until the redirection URL redirects them to a malicious sites. A server will respond to a client request with the `img` and `script` elements when accessing the compromised URL. However the `iframe` tag injected by an attacker is hidden, and thus a client will be redirected without knowing they have left the compromised site. This is due to the `iframe` element being set in an invisible state, which also greatly impacts the ability of existing web-crawlers to detect when a redirection has occurred.

2.1.3 Clickjacking

Clickjacking is a method that allows an attacker to manipulate transparent layers of a webpage to trick a user into clicking a redirection link as opposed to a visible webpage element [17]. Given the simplicity of implementing this method as well as the wide range of potential implementations, it is a common way to trigger unwanted redirections. Typically, a malicious actor will access the source code of a compromised URL or will create their own site disguised as a benign domain. Within the source code, invisible HTML elements are then created around the webpage that trigger a redirection link when pressed by a user. For example, if a user was trying to play an embedded video, a layer may be placed in front of the media player to trigger a redirection to a malicious site. This method is also commonly implemented within advertising elements of a webpage. Vulnerable embedded advertisement can be a target to attackers looking to phish private information from users.

2.2 Existing Methods

Given the rapid growth in internet usage coupled with the extreme global costs of malware ($17,700 a minutes [5]), many detection and prevention methods exist with the hope of reducing the number of redirection attacks.

2.2.1 Prevention

Open redirection attacks were amongst the earliest used by malicious actors due to the many flaws and vulnerabilities in best coding practices. As a result, more emphasis has been placed on secure programming as well as moving away from older content management systems and design practices. In particular, developers are encouraged to whitelist redirection URLs to ensure that parameter manipulation is not possible. Where whitelisting is not feasible, filtering based on protocol handlers is common practice. This is where a regex such as `^ https://` is used to ensure a user is redirecting to a webpage and not a JavaScript handler [18]. Furthermore, developers have moved away from implementing the redirection function at all, and instead tend to replace the feature with direct links to target URLs. Whilst the functionality is still used for authentication, the number of attack vectors has reduced significantly in many modern web pages. Additonally, a server-side list of permitted URLs is also becoming more common, making it more difficult for attackers to modify domain whitelists.

Client and server-side prevention methods for clickjacking attacks have shown to improve the rate of occurrence substantially [19]. Client-side methods include the use of browser extensions to prevent users from clicking invisible page elements or embedded documents. Examples of such technology include NoScript and NoClickjack, which are free plugins that block users from pressing invisible elements whilst not interfering with operations of legitimate `iframes` [20][21]. Server-side prevention methods for clickjacking include defensive code in a websites UI, which ensures that the necessary frame always remains at the top-most level window. Also, a frame killer is often used to prevent malicious web pages from being displayed. This prevention technique works by preventing any frames from loading that do not have explicit permission from a whitelist of URLs.

2.2.2 Detection

Despite advancements in prevention techniques, a large majority of users and websites still heavily rely on detection methods to prevent malicious redirections. However, malicious redirection detection techniques have historically struggled to withstand the growth in complexity of attack methods. A common security measure used to detect malicious redirections is to lookup a link against a blacklist of known malicious sites. A blacklist is typically generated through a botnet that scans the internet for sites with malicious code, plugins or certain IP addresses and registrars [22]. Using the list of flagged web pages, search engine providers will prevent or warn users from accessing a potentially malicious site. Currently, Google blocks around 10,000 harmful sites per day [23]. Despite the benefits of blacklists, the rate at which new malicious domains are generated impacts their performance greatly. Often, malicious sites are active for hours or days before being detected and blocked [22]. Furthermore, flagged sites may not be blocked from site-to-site redirections, and instead only prevent access directly from search results.

As the effectiveness of blacklists decline, new detection techniques have been explored with the intent of preventing malicious redirections. Machine learning techniques have started to be incorporated into redirection detection systems, with most showing strong results compared to traditional methods [16]. Different implementations extract various types of information using machine learning algorithms. Many models focus on either URL features or HTML source code elements to classify a website's nature. As will be discussed further in **Chapter 3**, these methods have severe limitations when used in isolation due to modern evasion techniques. As a result, there is yet to be an implementation recognised as the standard for machine learning based redirection detection.

Chapter 3

Related Work

The following research articles are closely aligned to the nature of this project. Each paper will be analysed based on their scope, similarities with this project, and advantages over other methods. Based on this analysis, a clear understanding of how previous work was used to guide this thesis will be established.

Beyond blacklists: learning to detect malicious web sites from suspicious URLs

This paper introduces a new malicious website detection method based on automated URL classification [24]. The authors use statistical methods to explore lexical and host-based features of webpage URLS. The dataset used in this study consists of 15,000 benign sources and 20,000 malicious sources containing either phishing forms or spam content [24]. Based on this dataset, appropriate features were extracted and used to train an SVM and Naive Bayes model.

The methodology followed by the authors was

- Collect dataset of malicious sites from multiple blacklists containing phishing links and benign links using the Yahoo search engine.
- Using this dataset, extract the various features from each link. These features include DNS NS record, hostname, path tokens, connection speed and quantity of characters in a URL.
- Train an SVM-lin and Naive Bayes model using the feature set.
- Tune false positives and negatives by altering the decision threshold for each classifier.

By using lexical and host-based features, the authors were able to predict a websites intent with an accuracy of 95% [24]. The study found that the models prioritised domain-based features such as length and path tokens, which implies that these are a more decisive way to distinguish malicious and benign websites. Despite the high total accuracy, the study found that the use of host-based features can lead to higher false positive rates. This was due to benign sites often being hosted at disreputable ISPs [24]. Furthermore, their model failed to detect malicious sites when they were a result of redirection services, used free hosting services, hosted in reputable geographical regions or were compromised benign sites. This is a large sample of data that would be overlooked by the models used in this study. In particular, as malicious redirection techniques have become more complex, this model would likely have a much higher false positive rate than at the time of release (2009).

Despite the age and small flaws identified in this study, it still provides a strong basis to guide future work. The results show that URL text features are the strongest way to distinguish between malicious and benign sites, as well as implying that additional text features would have enhanced model performance. Furthermore, the work highlights the considerations that must be taken when collecting data, including the potential bias that can be caused by using one source of data. Additionally, as evasion techniques have become more complex, the use of URL features to train classifier models is becoming less favourable. This is due to the use of URL shortening services by attackers, which increases the false positive rate of similar models significantly.

Detecting Malicious Web Links and Identifying Their Attack Types

This 2011 study used a variety of discriminative features like DNS information, textual properties, webpage contents and link structures to develop a binary classifier to detect malicious URLs [25]. The key findings of the study are as follows

- URL lexical features are effective at detecting phishing sites but do a poor job of detecting sites with malware since these pages do not differ in textual patterns by class. On the contrary, link popularity features were the strongest at detecting any type of malicious webpage. Additionally, webpage content features were particularly useful at identifying malware, as most sites would have malicious tags or scripts within their source code.

- Using an SVM model on a large dataset resulted in a total accuracy of 98.2%, as well as a 98.9% true positive rate and 99.2% true negative rate. However, when the size of the dataset was reduced, the true positive rate decreased by over 1.5%.

- False positives were typically caused if a benign URL had a low link popularity, no content on the webpage, contained a brand name keyword despite not being related or had an abnormal URL token.

- False negatives were hosted on popular social networking sites with a high link popularity as well as most hosted sites being benign. Also, URLs that used evasion techniques or had similar features to benign URLs were falsely classified.

Based on the above findings, it is evident that the authors effectively utilised machine learning models to classify malicious URLs. Considering the time of release, the models still hold considerable relevancy in the present context. However, the possibilities for evasion have certainly increased since these findings were published. For example, the use of more complex `iframe` redirections and URL shortening services render URL lexical features useless. Furthermore, Internationalized Domain Names (IDN) spoofing can be used to evade the outlined methods [25].

Despite the known evasion techniques for this method, the findings of this paper confirm that machine learning classifiers can produce fast website classification results. Whilst not quite real-time, the use of machine learning meant that individual sites can be labelled much faster than traditional techniques. Also, the paper is highly relevant to this project as it highlights the strengths of using webpage contents as a classifier feature for sites with malware. Thus, the findings of this study will provide a strong basis for the development of features as well as suitable models for training webpage contents.

CNN Based Malicious Website Detection by Invalidating Multiple Web Spams

This study looked at how a Convolutional Neural Network can be used to detect malicious websites based on a screenshot of the webpage [16]. The authors identified that modern evasion techniques are becoming too complex for existing solutions to cope. For example, the use of `iframe` redirections render many crawlers useless as they fail to detect malicious content. Thus, it was determined that a deep learning model on a screenshot of a webpage would solve this issue. The key findings of the report are as follows

- Using screenshots of a webpage makes it impossible for attackers to circumnavigate crawlers using hidden `iframe` spam.

- The CNN model worked optimally on a large dataset with similar nature web pages.

- From a test sample of 100,000 web pages, the CNN model had a total accuracy of 93.55%.
- Using a low learning rate, the optimal number of epochs before the model started to overfit the data was 51. This implies the training times for such complex models is high, however validation times are typically minuscule.

In addition the above findings, the paper also compared a series of machine learning classifiers using HTML source code to their CNN models. The results of which are shown in the below figure.

Description	**Method**	**Recall / TPR**	**FPR**
HTML Source	Naive Bayes	0.7502	0.1747
Code	SVM	0.7417	0.1033
Screenshots	CNN	0.9361	0.0531

Table 3.1: Comparison of different malicious detection algorithms

As can be seen in the above figure, the true positive rate (TPR) of the CNN model is significantly higher than the text-based models. This is due to the use of complex spam techniques to avoid detection of crawler bots. The CNN model however did not have this issue, as it is not possible for attackers to implement the same type of spam techniques. This finding highlights the viability of image classification to avoid evasion, and thus will guide the methodology of this thesis. The model parameters explored in this study also provide deep insight for the course of this project. In particular, they highlight the benefits of data augmentation to improve model performance and explore optimal techniques for a webpage dataset.

Despite the broad comparisons between text and image models, the paper does not provide a more detailed statistical analysis. However, the false positive results of both models are shown to be caused by different anomalies. As opposed to the use of spam techniques impacting the text models, the CNN model was mostly impacted by incorrectly loaded web pages or designs that did not align with a sites nature [16]. As a result, further investigation is required to determine whether the results of a system are more accurate if image classification is used in isolation, or in conjunction with text features. Given that there is minimal overlap between errors, a hybrid model may seem optimal. This will be explored further in the project, alongside ways to improve the text and image classification results presented in this study.

Chapter 4

Proposed Approach

4.1 Methodology

In order to achieve the overarching goals of this project, the following methodology was used. This methodology ensured that the outcomes of the project aligned with the goals, and also provides a way for future researchers to replicate the results. The below figure visualises the various steps that form the framework for this thesis.

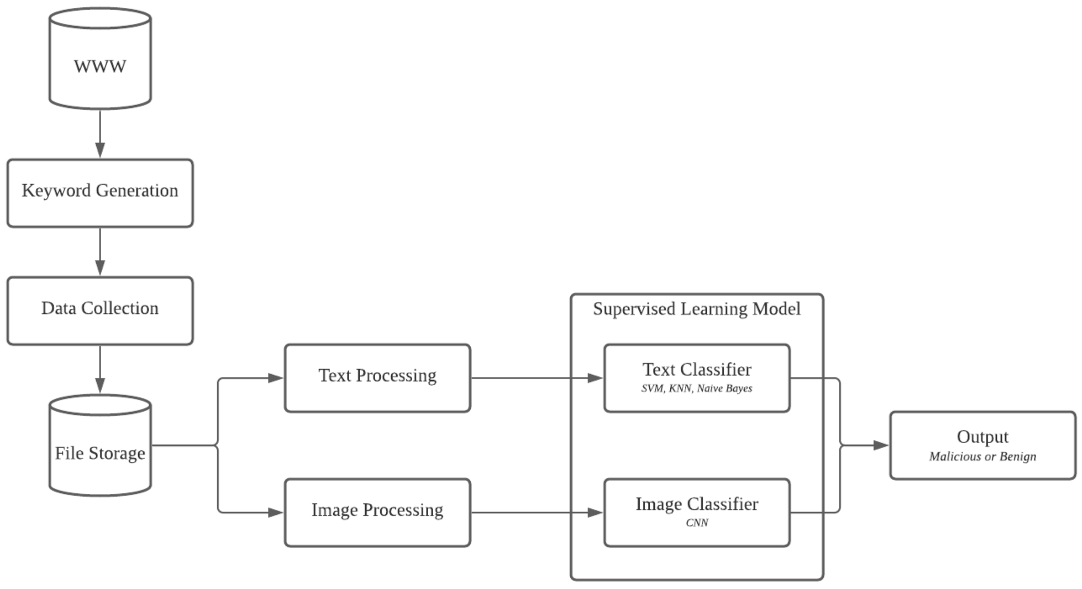


Figure 4.1: Project framework

For each step of this framework, a series of repeatable processes were followed and have been outlined in subsequent sections of this report. This ensures that future research in the field can replicate project outcomes and build on the methods used. Additionally, the framework divided each step into its own modular process, which ensured that the completed work was of the highest quality. An overview of each step used in the framework is as follows:

- **Keyword generation:** This process generated a list of thirty keywords that have a higher probability of leading to malicious search results. This list was based off prior research into dangerous search terms, and mainly targeted torrent, illegal streaming and file sharing websites.
- **Data collection:** This step involved using a custom `selenium` web-crawler script to generate a list of URLs based on the top search engine results for each keyword. The WayBack Machine Archive [26] and Google search engine [23] were used to generate URLs from the keyword list. Each URL was then visited by another `selenium` web-crawler script, where all `href` links were saved. These were then visited by the script and the user-facing text and a screenshot were saved to local storage alongside the source and destination URLs.
- **Text processing:** The raw text extracted was preprocessed to ensure the effect of a classifier was maximised. This step involved standardising the text corpus by converting all letters to lowercase as well as removing numbers, special characters and stop words. Furthermore, a TF-IDF filter method was used to extract only the most valuable words in the corpus to reduce model complexity and improve overall results.
- **Image processing:** The image dataset had to be standardised to ensure the deep learning model treated all samples with equal weight. Thus the RGB color scales were shifted from a [0,255] scale to a [0,1] scale, and all images were resized to 180x180 dimensions.
- **Classifier training and testing:** This process involved training and testing multiple text classifiers as well as a CNN model on image data. The text classifiers were trained in isolation using a 70/30 train/test split of data. The parameters of each model were optimised based on the feature set. The performance was then validated using a confusion matrix and accuracy metrics. By comparing these figures, the best models were selected and used to construct a final prototype.

Given the final implementation was designed to be representative of a working product, software engineering principles were followed in the development of this system. Code was produced to demonstrate how a user would interact with the classifiers while browsing the internet. Additionally, proper engineering principles were followed throughout the data collection and model development process. Python was selected as the software language for this project given its powerful machine learning libraries and ability to easily modify and visualise large amounts of data.

4.2 Keyword Generation

Keyword generation was the process of producing a list of approximately thirty potentially malicious search terms to be used for data collection. These keywords would then be used as input for the web-crawler, with the intention of resulting in a balanced set of malicious and benign URLs. In order to comprise a list, a variety of resources were used to find terms that have a higher change of leading to compromised domains. An annual report by McAfee produces a list of the web's most dangerous search terms based on the likelihood of finding a site with malware [27]. This report has been published over the last 15 years, and as a result shows how dangerous keywords have changed over time. In respect to this project, the historical data was particularly useful given the WayBack Machine was being used in the initial web-crawler. The WayBack Machine stores archived web pages, and as a result, modern dangerous search terms may not always be the most relevant. Consequently, a combination of older and recently identified dangerous search terms were used when forming the keyword list. This ensured that the results from the WayBack Machine would still have relevance given the potential age of search results.

In conjunction with the reports produced by McAfee, additional studies were used to assist with keyword generation. A study into search-redirection attacks found that nearly 30% of redirections from pharmacy related search terms led to sites with malicious content [28]. Thus, these were also incorporated into the final list of keywords. Furthermore, a study into malicious search terms found that keywords that target downloadable material are significantly more likely to result in a compromised domain [29]. This includes torrents, games and music downloads. Consequently, phrases that incorporated these terms were also used in the list of keywords.

The initial list of keywords used as input into the web-crawler were as follows:

torrent website	drugs online	adobe flash games
free movies	online coupons free	bitcoin casino
free downloads	codeine no prescription	cheat codes
pirated download	cialis no prescription	torrent games
christmas discounts	online pharmacy	free music
torrents	casino	free ufc stream
celebrity downloads	crypto casino	mma streams
free gambling sites	pharmacy	sports streams free
free music downloads	online prescription	celebrity torrents
amazing deals online	free games	boxing stream free

Table 4.1: Keywords used to generate the dataset

4.3 Data Collection

The data collection process incorporated a series of steps resulting in a complete dataset of webpage text and image features. Using the list of dangerous search phrases as input, a web-crawler was developed to interact with search engines in order to produce a list of domains. These domains were used as input in a separate script, which would save and visit any interactive link elements. The web pages would then be saved based on the content visible to a user. Based on the content extracted from each webpage, a manual labelling process was then followed to produce a complete dataset to be used by the binary classification models.

An overview of the data collection process can be seen below:

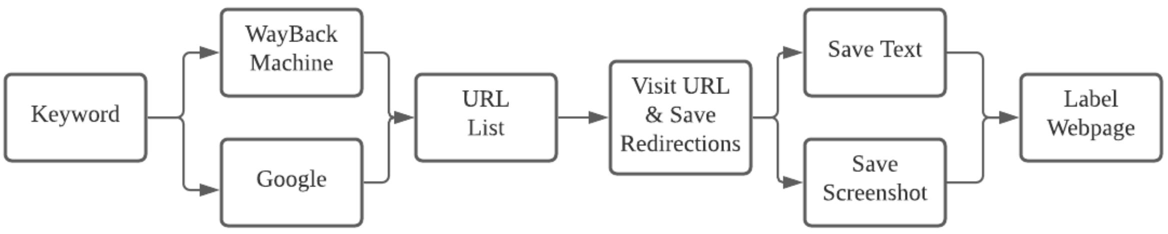


Figure 4.2: Data collection process

Despite the clarity provided by the framework, alterations were required based on the results first obtained. In particular, the way in which domains were accessed was altered to enhance the quality of search results (refer to **Section 4.3.4**). In total, the above framework was repeated two times after the initial process to ensure the quality of the dataset did not impact the model results.

4.3.1 URL Generation

The first step of the data collection process was to generate a list of domains from search results. The list of keywords was used as input for a Python web-crawler that interacted with the WayBack Machine internet archive [26]. The WayBack Machine archive is a search engine that collates website snapshots throughout the history of the internet. Based on a search term, the WayBack Machine can display a range of results from modern websites to sites over ten years old. The benefit of using this tool is that a wide range of search results can be produced based off a single term. Additionally, the security features in-place for this search engine are minimal when compared to common tools such as Google and Bing. These tools typically implement blacklists to block users from accessing any flagged sites. As a result, if these tools were used then the quantity of malicious data would likely be impacted.

The web-crawler utilised the selenium Python library to automate the process of searching a keyword and saving the results. In order to replicate human-browser interaction, the selenium package must have an executable driver located somewhere in the file-system. For this process, the Chromedriver executable was used. Once initialised, the selenium browser instance can be used to interact with any webpage as if a user was in control. Additionally, the source code of any webpage can be accessed, and specific elements can be called and stored in a Python data structure.

A valid WayBack Machine search URL was comprised by adding each keyword to the prefix `https://web.archive.org/web/*/`. The first page of 20 search terms could then be extracted via the `result-list` class located within the source code. These results were in a URL format, which enabled the script to simply store them in a `.txt` file for future use. A snippet of the code used to extract these URLs is shown below.

```
def create_webdriver():
    chromedriver_path = 'Chromedriver/chromedriver'
    return webdriver.Chrome(executable_path=chromedriver_path)

def get_urls(url):
    browser = create_webdriver()
    browser.get(url)
    timeout = 5
    try:
        element_present = EC.presence_of_element_located((
            By.CLASS_NAME, 'result-list'))
        WebDriverWait(browser, timeout).until(element_present)
        elements = browser.find_elements_by_class_name(
            "result-item-heading")
        return elements
    except TimeoutException:
        print("Timed out waiting for page to load")
        return None
...
```

The result of using this method to extract links was a list of 600 URLs that could potentially lead to malicious redirections. After removing overlapping URLs, the total number of links saved was 574. This overlap was to be expected given the ranking algorithm used by the WayBack Machine as well as the similarities between many of the search terms used.

Further Iterations

After completing all steps of the data collection process, it was evident that the sample size of quality data wasn't to the standard required for a meaningful outcome. A large portion of URLs appeared to broken or no longer in use. As a result, the potential dataset size was significantly reduced, which would have had adverse effects on model performance down the line. Thus, the URL generation process was repeated following a different method with the aims of improving the quality of results.

Whilst the WayBack Machine did successfully produce a small sample of malicious web pages, most results were either unusable or benign. After completing the steps outlined in **Section 4.3.3**, it was found that only 8% of all redirections were malicious, and the total labelled dataset was 15% of the initial sample size collected from the WayBack Machine. This is indicative of a poor quality dataset with an uneven distribution between categorical data. In order for a binary classifier to work effectively, it is important to have an even spread amongst categories to ensure the model has an optimal amount of training data. Thus, a new method for URL generation was proposed that utilised the Google search engine. Initially, this was being avoided based on the security measures in-place. However, it was found that by using the `googlesearch` library, security preferences could be disabled [30]. Thus, this was used to produce an additional list of domains based on the same keyword list.

The `googlesearch` library is an open-source Python library that allows one to programmatically interact with the Google search engine via their API. Additionally, it supports lists of inputs and an exact number of result URLs can be determined within it's methods. Thus, the use of this library was found to be incredibly simple and efficient. A code snippet of the library being used to extract URLs can be seen below.

```
def get_urls(keywords):
    f = open("url_list.txt", "a")
    for keyword in keywords:
        for url in search(
            keyword, tld="com",
            safe='off', num=25,
            stop=50, pause=20):
            f.write(url + '\n')
    f.close()
    return "Success"
...
```

As highlighted by the code above, the implementation was made very simple by the `googlesearch` library. The parameter values were configured to maximise the chance of search results being malicious. This included using a `.com` top level domain, and restricting safe mode. The final list of URLs produced from this script had a total length of 750, with little to no overlap. Additionally, based on initial observations, it was clear that the quality of web pages had significantly increased and the number of broken or unused links was minimal. Given this new list of domains, it was thus appropriate to extract relevant features to be used for classifier training and validation.

4.3.2 Text & Image Extraction

Using the complete list of generated domains, the text and image contents of all redirections needed to be stored in a dataset to be labelled at a later time. The selenium Python library was again used to visit each link, as well as to follow all redirection paths and save the destination image and text content. A Python program was developed to make this task as modular as possible. The program contained two key files, which were:

- `follow_urls.py`: This file stores the input list of URLs into an array. This array is iterated over, where the script then calls an instance of a `LinkFollower` class, using a URL as a constructor parameter.

- `link_follower.py`: This file initialises a `LinkFollower` instance. On construction, a selenium browser session is created, and a series of browser profiles are configured. These ensure that the result will replicate the exact interactions a user would experience. The instance then visits the original link and saves all `href` tag elements. These `href` links are opened using the same browser session, where the heading and paragraph HTML elements are added to a `.csv` file. Finally, a screenshot is taken of the webpage and is assigned a random large integer to avoid collisions. This integer is stored in the `.csv` file alongside the text data and the image is stored in a seperate directory using the integer as a filename.

As is evident from the above descriptions, the `link_follower.py` replicated website redirections and stored key data in local storage. It was determined that `href` tags would be the most effective way to replicate redirections as they are abundant on most websites. Whilst malicious redirections can also be triggered through `iframe` redirection attacks, it was infeasible to use these methods to generate a large dataset. Furthermore, the destination of all redirection attacks are of the same nature, and thus the method of redirection itself does not impact the quality of data. Hence, redirections via `href` tags was the method this program was trying to find and record. Additionally, by storing the origin and destination URLs, it was clear to visualise whether a malicious redirection had taken place. Typically, the URL of malicious sites will have no correlation to the origin domain, and thus makes it easy to determine the nature of a redirection.

The text that was extracted from the destination link was all user-facing. These are all the words contained within the heading and paragraph HTML tag elements. Given a text classification model uses individual words as input as opposed to complete phrases, the order in which these were extracted had no relevance. Thus, the simplest process was to find all elements and create one large string to store in the unlabelled dataset.

The final unlabelled dataset consisted of roughly 30,000 web pages collected via the `href` tag elements of the origin URL list. Additionally, each webpage stored the origin URL, text content and image index to be used to inform the labelling process.

4.3.3 Data Labelling

Labelling the dataset was a time-consuming and challenging process that required a clear framework for a successful outcome. Originally, multiple semi-supervised models were going to be used to help automate the process. These models would use a range of features including text, image, and URL lexical features to assign labels to the data. Then, a manual review could be taken to verify the results, or to alter the labels and thus further refine the model in the process. However, given the quality of the initial data, using this method was not desirable. This was due to the excessive number of broken links and web pages with no content. Furthermore, given that `href` elements were used, there were instances where one URL would produce thousands of redirection links. This introduces potential bias into the model if not addressed, as often many of these destination pages share very similar features. As a result, the dataset can be diluted by a single style of webpage, which has adverse effects on classification models.

Based on the flaws of the dataset, it was determined that manual labelling was going to produce the best outcome. Whilst the process was not as efficient as desired, it ensured that the quality of data remained consistent throughout the dataset. It also meant that the accuracy of labels was high assuming a methodical process was followed. As a result, it was important to clearly define what constitutes a malicious redirection. Thus, the below guidelines were created to assist with the labelling process.

A malicious redirection has taken place if one of the following has occurred:

- The destination URL contains links to illegal or unverified downloads.
- The contents of the destination URL does not align with the link itself and does not match the nature of the source website.
- The destination URL offers services or downloadable content from an unverified company.

The above definitions were formed based on the background research of this project. Whilst it may seem that these can lead to false positives, this is not necessarily a bad thing given the focus of the project. It is much safer to wrongly classify a site as malicious than benign. Furthermore, given this process can be adapted to create a user-facing product, false negatives can have significant impacts on the success of the product. Thus, whilst these definitions may lead to some false classifications, as long as the number of false negatives is near zero, there is still valid use for the model.

Another key point that was considered when manually labelling data was the possibility for human error. Given the sheer size of the raw dataset, it was essential to have a clear method to ensure labels remained accurate. Thus, the following method was followed throughout the process:

- Iterate through and label 100 rows of the dataset by observing the screenshot and text features. Remove poor quality data from dataset.
- Re-iterate over the 100 labels and confirm they are accurate.
- Add new batch of labels to final dataset.

By using the above method, the completed labels appeared to be accurate and the amount of misclassification was minimal. Additionally, the size of the dataset was significantly reduced as poor quality data was removed. From the original 30,000 rows of unlabelled data, the final dataset contained 10,934 websites. Of these, 7,328 were benign in nature, and 3,606 were malicious. An example of the final dataset can be seen in the below figure.

from_url	**to_url**	**text_content**	**image_index**	**label**
http://isohunt.com/	https://www.w...	Watch Summer ...	2819304...	benign
http://amazingjak...	https://www.livein...	popular news site...	8374950...	malicious
http://torrentsp...	https://www.motio...	THE LATEST ...	4729474...	benign
http://imusicdow.../	https://qy1815.com...	Your current IP: ...	3646372...	malicious

Table 4.2: Example rows of labelled dataset

4.4 Model Development

4.4.1 Text Classification

A core component of the produced work is the text classification model. This is used on the user-facing text components of a labelled web-page. Whilst other implementations often incorporate other webpage elements into their text classification models, the decision to use only user-facing components was due to the model being used in conjunction with image data. Given this hybrid model, attempts at evading detection via hiding malicious content within `iframes` is thwarted. Thus, the simplicity and efficiency of using HTML text as input for a classification model far outweighs the small odds of both models being evaded by an attacker.

Given the simplicity of implementing text models, the decision was made to build multiple classifiers and compare results. Thus, a naive Bayes, support vector machine, and k-nearest neighbour model was developed using the text data. This ensured that a comprehensive analysis could be undertaken and the strongest possible model could be used for future implementations.

Text Processing

In order for a text classification model to work effectively, redundant features from the corpus must be removed. Text preprocessing aims to bring the text corpus into a predictable and analysable form, whilst also removing waste that impacts model efficiency and effectiveness. For this project, the dataset was loaded into a `pandas` dataframe within Python and the `nltk` library was used to perform a series of preprocessing steps. Prior to following a standard text preprocessing method, a large part of the corpus had to be translated into English. Given that the search scripts were not location specific, a lot of website results were in a foreign language. In order to keep the corpus of text consistent and minimise complexity, it was essential to ensure all of the data was in the same language. Thus, a script was developed that made use of the `google_translator` library. This library allows one to make calls to the Google Translate API through Python. As a result, the process can be automated, which given the size of the dataset was essential. Another benefit of this library is that it detects the language prior to translation. Since there was such a broad range of languages in the initial dataset, this simplified the process significantly.

Once the text corpus was in English, the preprocessing techniques used as well as their order was as follows:

- **Convert text to lowercase:** The `string` library in Python has a `lower()` method that converts all characters of a string to lowercase. This is beneficial as it halves the number of potential characters a model has to use as input. Consequently, reducing this noise increases model efficiency and helps standardise the data regardless of its source.
- **Remove numbers:** The `re` library in Python enables the use of regular expressions. Thus, the `sub` method can be used to substitute any numeric values for a blank string.
- **Remove special characters:** The `string` library can be used to replace all punctuation with an empty character. This again standardises the text corpus and significantly reduces total complexity.

- **Remove trailing and leading white space:** Often, a word or phrase can have whitespace attached to it. Thus, if the same word is compared but one has whitespace on either side, a machine learning model would not interpret them as the same. The `.strip()` removes all trailing and leading white space from a string, and was thus used on all rows of data.

- **Remove stop words from the corpus:** The `nltk` library provides a corpus of stop words. A stop word is a commonly used word that adds no value or information to a sentence. Using Python, it is relatively simple to extract and remove all instances of stop words from a string using the `nltk` library. Prior to doing so, each row of data was first tokenised such that a string was converted into an array of individual words.

- **Extract top words in the corpus:** Using a TF-IDF filter method, it is possible to extract the top percentile of words within the entire text corpus. A TF-IDF filter method is a statistical measure to evaluate how relevant a word is to a corpus of text [31]. It calculates the term frequency of a word, which is the number of time it appears in the document. Furthermore, the method determines the inverse document frequency of a word, which is the measure of how significant it is in respect to the corpus. This is determined by calculating the log of the document frequency. The document frequency of a word in this dataset is the total number of rows divided by the number of rows containing a given word. In Python, this can be implemented simply using the `sklearn.feature_extraction` library. This will refine the entire corpus to only contain words above a certain calculated TF-IDF threshold value.

The result of these preprocessing steps was a corpus of text with significantly less data. Initially, the corpus contained 410,432 total words. After preprocessing, this was reduced to 184,902. Furthermore, as is explicated by the table below, the most common words in the corpus shifted dramatically.

Before	**After**
the	online
to	torrent
and	vpn
of	use
a	best
you	casino
is	website
in	free
your	sites
that	information

Table 4.3: Most common words in text corpus relative to preprocessing steps

As can be seen from the figure above, prior to preprocessing, the most frequent words were entirely stop words. However, after the text had been filtered, this changed to a series of words that aligned closely with the original keyword list. Thus, the results of the model were significantly benefited by these steps, as a more accurate representation of a websites nature was being considered based on meaningful text as opposed to terms found amongst most websites.

Partitioning the Dataset

In order to train and validate the models, a normal train test split was used. The optimal ratio was determined to be 30%, meaning that 30% of the total dataset would be used to validate model accuracy. This was determined given the total dataset size and number of features within the text corpus. Given the large corpus, it was important to use enough data to allow the models to see as many edge cases as possible as well as learn trends in the feature set. However, it was also important to use enough of the data to validate the model accuracy to ensure that a meaningful outcome could be achieved. Thus, a 70/30 split was deemed as a fair compromise given the nature of the dataset.

The data was split using the `sklearn` library. The `train_test_split` method divides the dataset randomly at a ratio of 0.7 based on a column value. In this case, the malicious and benign labels were used to determine the ratio. These were then stored in separate `numpy` arrays, and used to train and validate the various text models.

Naive Bayes

A naive Bayes model is a simple probabilistic classifier based on applying Bayes theorem with strong independence assumptions between features [32]. It has been one of the most popular machine learning techniques in recent years, and is especially favourable for text classification problems. The simplicity and reasonable performance make the model desirable, as well as its ability to produce accurate outcomes with a relatively small sample of training data.

In existing naive Bayes text classifiers, a document is considered a binary feature vector representing whether each word is present or absent [32]. This type of model is called a multinomial naive Bayes, and has been heavily leveraged in existing spam classification implementations. The model assumes all features are independent of each other given the class context. With a multinomial event model, feature vectors represent the frequencies in which events have been generated by a multinomial [32]. The model can be defined by the formula,

$$P(x \mid C_k) = \frac{(\sum_{i=1}^{n} x_i)!}{\prod_{i=1}^{n} x_i!} \prod_{i-1}^{n} p_{ki} \cdot {}^{x_i}$$

This essentially states that the probability of a feature vector `x` belonging to a class `C` is equal to the sum of occurrences over the product of occurrence, multiplied by the probability p_i that some event `i` occurs. Based on this formula, if a specific class and feature value don't occur together in the training data, then the frequency-based probability estimate is zero [32]. This can lead to problematic outcomes when the training sample is exceedingly small. A small training sample means that if a feature value appears in a class in the validation set but not the training set, the assumed probability is zero. This can lead to large false negative rates, which in the context of detecting malicious redirections is a potentially dangerous outcome.

The multinomial naive Bayes model can also be impacted by its rough parameter estimations. In this model, word probabilities for a class are determined by the likelihood of occurrence in all positive training documents [32]. Thus, when all training vectors for either class are combined into a single corpus, parameter estimation is influenced by the length of each individual vector. This means that web pages with more words will have a higher impact on the results, as more words participate in estimations. When looking at the shape of the dataset for this project, the variation in lengths of text is significant. The longest text documents contain many hundreds of words, whilst the smallest contain only a handful. Thus, model bias may significantly effect model outcomes.

For this implementation, the `sklearn.naive_bayes` library was used. The MultinomialNB class is an implementation of a multinomial naive Bayes model. The `numpy` arrays containing training data and labels were used to fit the model. In terms of parameters, the default values were used. Thus, no smoothing parameter was set and uniform priors were used.

SVM

A support vector machine (SVM) model is a supervised learning model that uses classification algorithms for binary classification problems [33]. Unlike naive Bayes, support vector machine models are non-probabilistic, meaning that given a set of training examples, an SVM algorithm will build a model that assigns validation data to one class or the other. It does this by constructing hyperplanes (decision boundaries) in a high-dimensional space to maximise the separation between two classes. A key advantage of using this algorithm is its high efficiency and performance when working with small sample sizes. Additionally, the algorithm is effective in high dimensional spaces and can still produce accurate results on sparse data. In terms of this project, this is necessary given that often features of the text corpus do not appear more than once. For example, a malicious website may contain content that significantly strays from the 'norm' defined by the dataset. As a result, these occurrences will likely be low, but given that the SVM algorithm is non-deterministic, the chances of this feature vector being correctly classified is still reasonable.

The SVM algorithm is also versatile as it has the ability to adapt different kernel functions based on the nature of a particular dataset. Thus, non-linear hyperplanes can also be fit to a feature-set at a relatively low computational complexity through the use of the kernel trick. This method performs non-linear transformations of the dot products between two points [33]. Non-linear kernels are typically useful when dealing with datasets with a small feature space. Thus, given the nature of this text dataset, a linear kernel implementation is favourable. This is because an SVM model is likely to overfit to a complex feature space when applying a non-linear kernel function. As the complexity of a feature space increases, non-linear hyperplanes will typically form small margins between classes. As a result, if the validation data set deviates from the training sample even slightly, the accuracy can be greatly impacted.

The implementation of the SVM algorithm for this project used the `sklearn.svm` library. The `SVC` class was utilised, which is an implementation of the SVM algorithm but for classification problems. Within this class is a series of parameters that must be specified when fitting a model to the training dataset. The modified parameters as well as their default values are as follows:

- **C** = 1.0: This is a regularisation parameter. The strength of the regularisation is inversely proportional to this value [34]. This parameter defines how much the model should avoid misclassifying each training example. Thus, for large C values, the optimisation will select a small margin hyperplane. Given the size of this feature space, the default C value will ensure that over-fitting does not occur. However, tuning this value is an effective way to determine the optimal amount of optimisation.
- **kernel** = linear: The kernel function used by the classifier. This defines the type of hyperplane fit to the dimensional space. As explained previously, a linear kernel is optimal given the dataset.
- **degree** = 1: This parameter defines the flexibility of the polynomial used to find the decision boundaries. A higher degree kernel yields a more flexible decision boundary, similar to changing the kernel type. As shown in **Section 5.1.2**, altering this value can have significant effects on the overall model accuracy.
- **gamma** = auto: The gamma parameter determines the amount of fit to the training dataset for non-linear decision boundaries. A higher gamma value results in a tighter fit to the training data.

KNN

K nearest neighbour classification is a non-parametric classification method that is highly suited to the dataset of this project. The algorithm assumes that features from the same class exist in close proximity within the feature space. Thus, the 'k' parameter defined in the model determines the number of neighbours for a single point to consider as similar. Based on this value, a decision boundary between classes is formed, which is then used to classify the validation set. The model has many benefits, including its simplicity and versatility in dealing with a range of problem sets. However, the algorithm can be impacted by a skewed class distribution. Given the slightly skewed shape of the dataset developed for this project, it is possible that results may not be optimal. The more frequent class, or in this case benign websites, will tend to dominate predictions as they are the most common neighbours given their quantity. As a result, false positives can be a likely occurrence if the value of 'k' is not optimal given the distribution of classes.

In order to determine the optimal 'k' parameter, the `sklearn.neighbors` library can be used to iterate through a large range of values. In doing so, the suitability of the model for the dataset can also be determined. The code used to train and text each value of 'k' can was as follows:

```
neighbours = np.arange(1,20,1)
for neighbour in neighbours:
    knn = KNeighborsClassifier(n_neighbors = neighbour)
    knn.fit(train_vectors, y_train)

    # Training Accuracy & Confusion Matrix
    knn_prediction = knn.predict(train_vectors)
    accuracy = knn.score(train_vectors, y_train)
    cm = confusion_matrix(y_train, knn_prediction)
...
```

Based on the strongest accuracy results from the above code, it is clear whether the KNN algorithm is suitable for the dataset. If a value of k = 1 appears to have the highest train and validation accuracy, then this is indicative of a model that has over-fit to the training sample. This case is an implementation of the nearest neighbour algorithm, which only considers the next closest neighbour when forming a decision boundary. If the validation data deviates even slightly from the training sample, the results of this model will be significantly impacted. This is because the decision boundary between classes that is formed from the training sample is as tight as possible, which means that any outliers or scarce features will likely be incorrectly classified.

4.4.2 Image Classification

Image Processing

Prior to building a classification model, it was essential to perform a series of preprocessing steps on the image dataset. Image preprocessing is an essential method that enhances the quality of dataset images in order to improve the analysis performed by a machine learning model [19]. Furthermore, when done correctly, image preprocessing should suppress unwanted distortions and enhance important features to help train a classifier. Additionally, for models such as convolutional neural networks, all images must be the same sized arrays. Furthermore, image processing can significantly decrease model training time and increase inference speed. When dealing with sizeable image datasets, this can lead to profound differences in training times. In the context of this project, whilst training time does not directly impact the final implementation, it does impact the speed at which alterations can be made. Thus, if the quality of the image dataset can be maintained whilst reducing complexity, preprocessing will always be of benefit to a classification model.

Prior to performing any preprocessing steps, the data first had to be restructured within the local directory it was being stored in. This was to simplify the process of loading train and test samples. Thus, the `.csv` dataset containing a websites label and image index was used as input for a Python script alongside the single image directory path. With this, the `os` library was used to find the location of each image within the main directory. These were then moved to a new folder depending on their label. The new folder structure had two folders instead of one; a folder for both malicious sites and benign sites. Then, the images were loaded into the Python notebook using the `pathlib` library. The `Path` object within this library allows a variable to be assigned as a pointer to a local directory. After this, the `list` method can be used to assign all images to an array variable. In doing so, both malicious and benign sets of images were loaded into a Python notebook, allowing preprocessing steps to then be performed. An example of the code used to store these directories within Python can be seen below.

```
import pathlib
data_dir = "/content/Updated Data/" # Directory path
data_dir = pathlib.Path(data_dir)

# Array of malicious images
malicious = list(data_dir.glob('malicious/*'))

# Array of benign images
benign = list(data_dir.glob('benign/*'))
...
```

Once the data was loaded, prior to performing preprocessing steps, data exploratory analysis was undertaken. First, examples of benign and malicious sites were displayed to ensure the data had been loaded correctly. It also provided an opportunity to compare features and see what preprocessing steps may be effective given the type of images. Below is an example of both a benign and malicious website from the dataset.

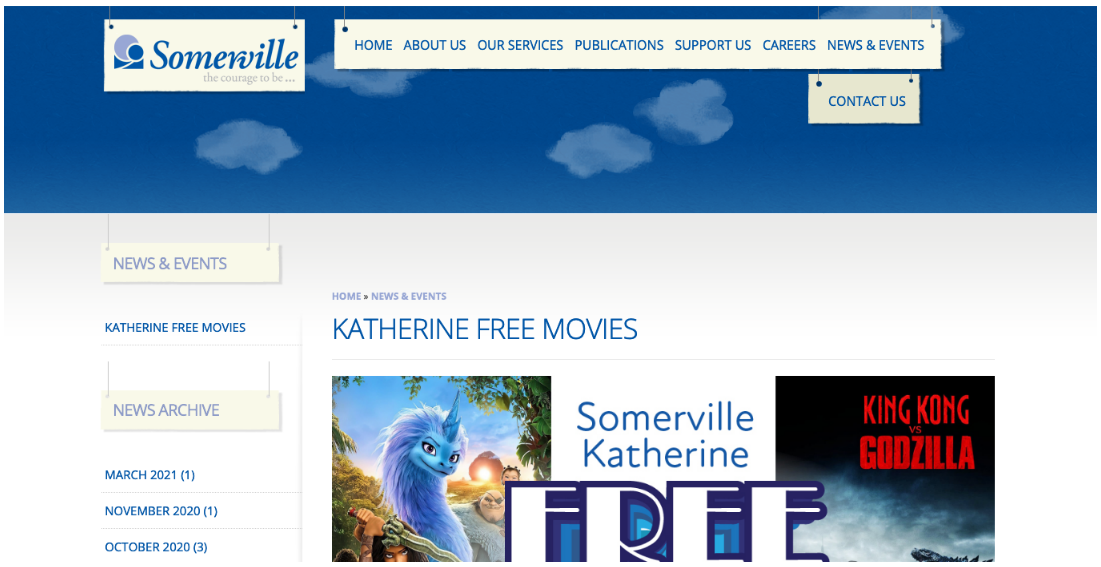
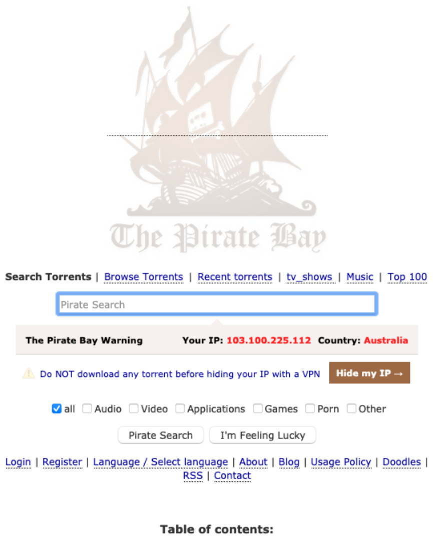


Figure 4.3: Example of benign website from dataset


The Pirate Bay
Search Torrents | Browse Torrents | Recent torrents | tv_shows | Music | Top 100
Pirate Search
Hide my IP →
all Audio Video Applications Games Porn Other
Pirate Search
I'm Feeling Lucky
Login | Register | Language / Select language | About | Blog | Usage Policy | Doodles | RSS | Contact

The Pirate Bay Warning Your IP: 103.100.225.112 Country: Australia

Do NOT download any torrent before hiding your IP with a VPN

Table of contents:

Figure 4.4: Example of malicious website from dataset

After this, the data was split into train and test partitions. Similar to the text classification models, the data was partitioned using a random 70/30 split. This provided enough data to train the model effectively whilst also providing a large enough sample to accurately validate results. Next, a series of malicious and benign sites were compared using a 3x3 grid of images. These were randomly pulled from the dataset, with the idea being to compare features and determine appropriate preprocessing steps. An example image matrix from the dataset can be seen below.

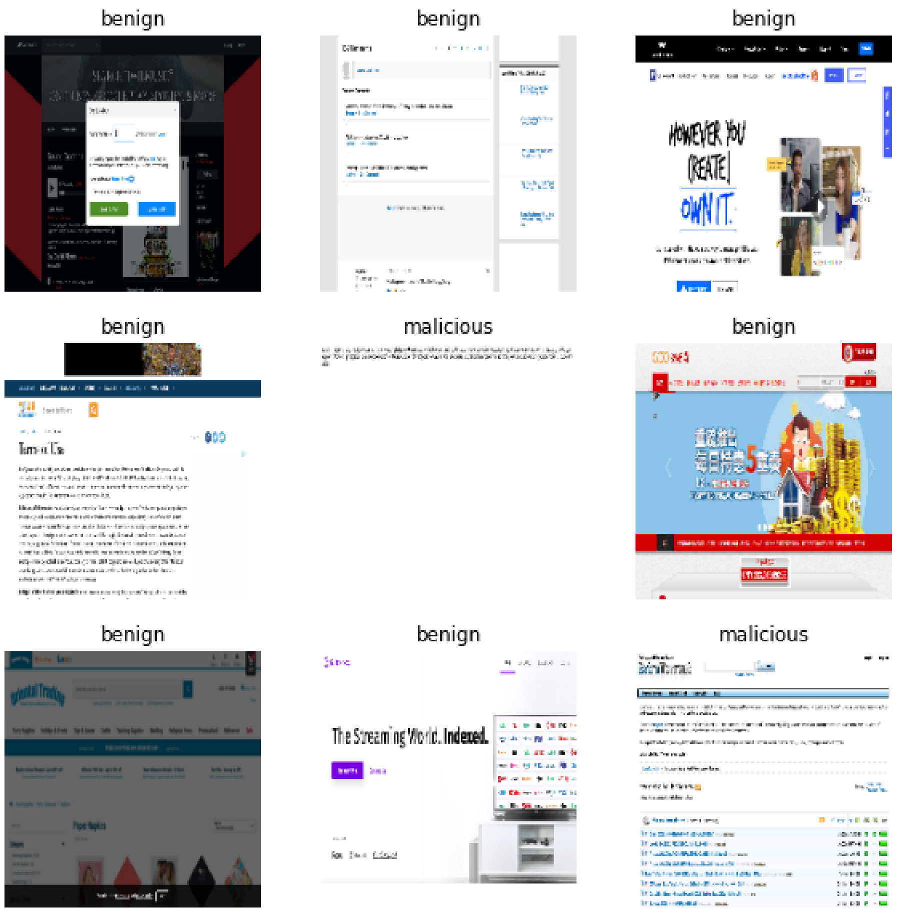


Figure 4.5: Comparison of images belonging to different classes

As can be seen from the above figure, there are discrete yet noticeable differences between both classes. This was particularly evident after repeating this step multiple times. Firstly, it was clear that benign sites typically had a more modern and 'professional' webpage layout. Additionally, benign websites often used larger and better quality images. These types of trends should be distinguishable by an appropriate machine learning model. Thus, this method of observation provided positive signs that there were indeed distinguishing features between malicious and benign web pages.

After initial data exploratory analysis, there were no noticeable discrepancies between images that needed to be resolved through preprocessing steps. As a result, a more standard series of preprocessing steps were followed with the intention of improving model complexity. Firstly, the RGB channel values were transformed from [0, 255] to a [0, 1] range. This used the `keras.layers.Rescaling` function. This is a `tenserflow` library that normalises the image data such that original RGB channel values are reflected on a [0, 1] scale. Finally, the images were resized to a 180px x 180px shape to help speed up the training process. If this value is too small, certain features may be ignored by a model, and if it is too high the epoch training times can be excessively long. Thus, it was found through trial and error that using images of 180x180 dimensions was optimal.

CNN

When dealing with image datasets for classification problems, deep learning has become standard practice due to it's high accuracy and flexibility. A convolutional neural network (CNN) is a deep learning algorithm that is known for its superior performance when handling image and audio data. As the name implies, the algorithm uses convolution as opposed to matrix multiplication in at least one layer [35]. The strengths of the model come from its ability to manually extract features from large samples of images. The convolution and pooling layers perform complex feature extraction by initially identifying small features and building on top of one-another. Using this dataset as an example, the first layer may detect specifically edges. The next may then identify shapes, where the following layer merges the two to make some inference about the feature. When done over many layers, the inferences made by the model can get very specific, and features unnoticeable to humans can be identified and classified by the model. An overview of a typical CNN architecture is as follows:

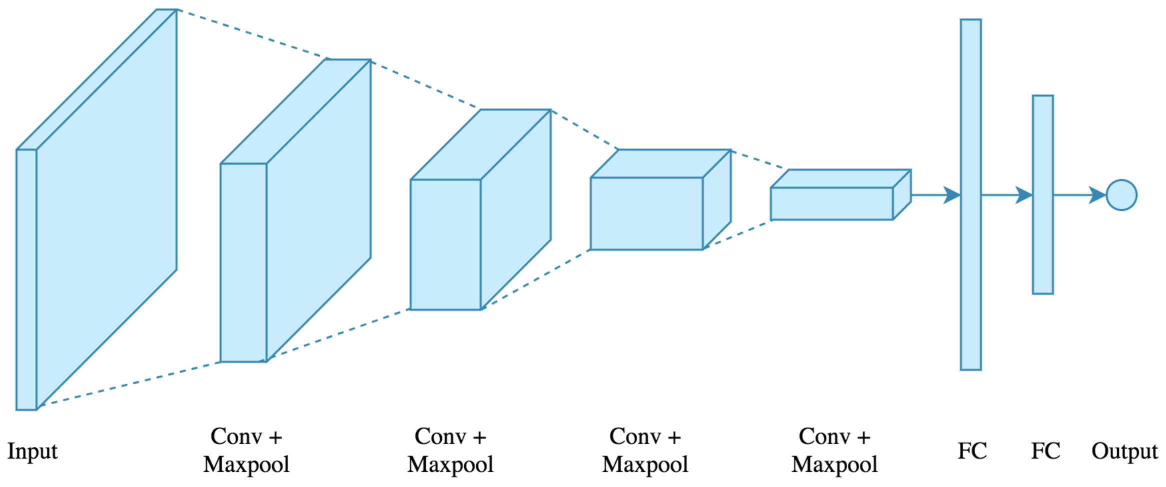


Figure 4.6: Overview of CNN model structure [36]

As can be seen in a typical CNN model structure, the convolutional layers perform a series of operations to merge sets of information. The layers use a convolution filter to produce a feature map from image data. Multiple convolutions are performed on a single input, with each one using a different filter and producing a distinct feature map. These are then stacked together to produce the final output of a single convolution. Pooling is then performed to reduce dimensionality by reducing the number of parameters in the feature mappings. This is then repeated for multiple convolution and pooling layers. Finally, fully connected layers flatten the output to a single vector to improve model performance [36].

The detection speed of CNN models is also a key advantage over other image classification processes. Despite long training times, running a single image through a trained model can produce an output in near real-time. In the context of this project, this was essential given the intended application of the model.

To implement a CNN classifier, the `tenserflow` library was used. The `Sequential` class involves adding layers to a model individually in a linear manner, from input to output. By using this, it is possible to specify the quantity of convolution and pooling layers for a CNN model. Thus, the `Sequential` model consisted of three convolution blocks with a max pooling layer in each of them. A fully connected layer was also added to the end of the `Sequential` model, as outlined in the diagram in **Figure 4.6**. An overview of each library used within the `Sequential` model implementation and their parameter values is as follows.

The `tf.keras.layers.Conv2d` object represents a 2-dimensional convolution layer in a CNN model. The parameters for this object as well as the chosen values are highlighted below. It should be noted that the object has many other parameters that were not configured in the final image classification model.

- `filters` = (16, 32, 64): This represents the number of output filters in the convolution. This value was doubled for each convolution layer. As the number of layers increase, the patterns get more complex within the model. Thus, larger combinations of patterns must be captured, meaning that the filter size must increase alongside the depth of layers.

- `kernel_size` = 3: The kernel parameter specifies the height and width of the convolution filter. A small size causes under-fitting whilst a large size can lead to over-fitting. A kernel size of 3x3 is a standard value that was altered throughout the testing phase of the project.

- `activation` = relu: The activation function of a convolution layer refers to how the weighted sums of inputs is transformed into an output. A relu activation function is the most commonly applied to deep learning problems given its efficiency and accuracy.

The `tf.keras.layers.MaxPooling2D` object reduces the dimensionality of the convolution outputs. For this implementation, no parameters were modified and the default values were used. The `tf.keras.layers.Dense` fully-connected layer was configured to have 128 units on top of it that are activated be a ReLU activation function. This is double the filter size of the final convolution layer, and ensures that large combinations of patterns are captured in the output.

After creating the model, it was compiled using an Adam optimiser and a sparse cross entropy loss function. The Adam algorithm optimised model performance by reducing the cost function associated with the model [36]. It is an efficient algorithm that works optimally with sparse data and large feature sets. This aligns with the nature of the image dataset, and thus was chosen as the best option. A sparse cross entropy loss function calculates the gradients which are used to update the weights of the model. Finally the model was trained over 10 epochs to determine the optimal number of iterations given the dataset.

4.5 Applying the Models

The purpose of this project was to go beyond simply building accurate models to predict malicious redirections, but to also implement a working demonstration to show how they can be applied in a meaningful way. Thus, a sizeable portion of the project was spent determining an appropriate way to implement a hybrid of a text and image model into a working prototype. A key consideration that had to be made when brainstorming potential solutions was how text and image models could be used in conjunction to produce a single output. Given the models have different strengths, there was likely going to be certain sites where classifications would not align between them. Thus, a conservative approach was taken when building a prototype. Given the context of this project, it is far safer to produce false positive results rather than false negatives. Therefore, it was determined based on the results of model training and testing that so long as one model outputs a malicious label, that site would be deemed malicious (see **Section 5.3** for a more detailed analysis).

Based on the notion that a conservative approach would be used, a specific implementation had to be designed to address shortcomings in previous work. Historically, botnets have been used as the means of classifying malicious sites using machine learning models [25] [37]. Typically, these systems will scan the net and access many URLs, where site information will then be fed as input into a machine learning model. If the model assigns a malicious label to a site, it will be added to a blacklist of domains that are then checked every time a user is redirected to a new link. Whilst this method is effective due to its efficiency and ability to deploy multiple bots at once, there are still major limitations that need to be addressed.

The key issue with this method is that the rate at which bots can detect malicious sites is far less than the rate new malicious sites are created. Thus, regardless of the accuracy of the machine learning models being used, it is likely that a user will encounter an undetected compromised URL. As a result, a solution is required that ensures all sites a user visits are analysed using the models.

Given that feeding test data through text and image classifiers is an efficient process, it is possible to determine the nature of a website in near real-time. The ideal solution to address flaws in previous work would be to use a browser extension to parse all visited domains through the trained classifiers. This would ensure that all discovered URLs are being checked, eliminating the common evasion technique used by attackers whereby new host links are regularly generated for compromised domains. Additionally, it is computationally inexpensive to run an instance of the trained models on a client's browser, and would result in an efficient data retrieval process. Finally, by using this process, it ensures that user-facing content is being checked by the models as opposed to dummy-content hiding the actual malicious content of a webpage. An overview of a complete implementation for the proposed system is as follows:

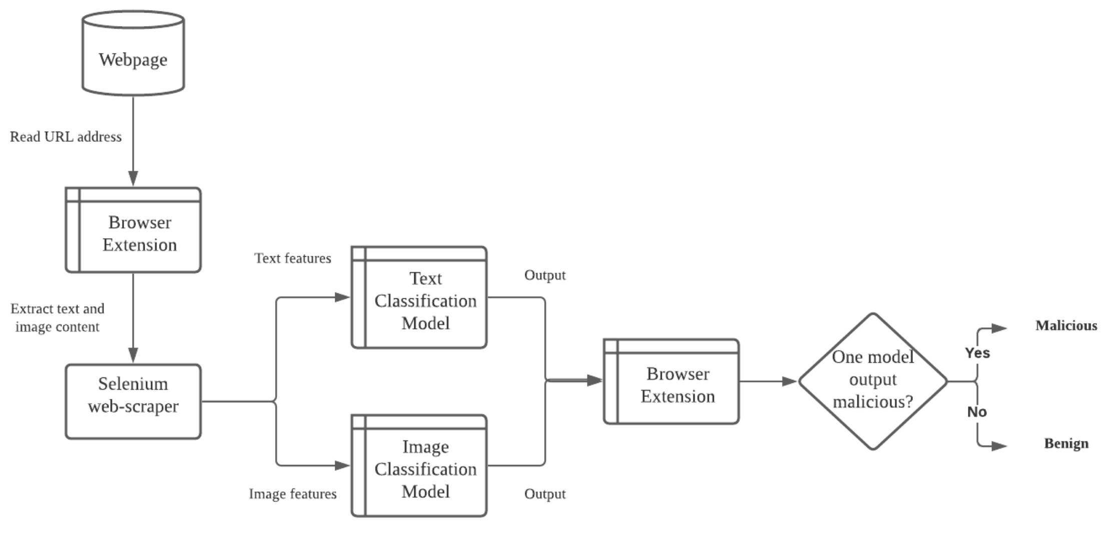


Figure 4.7: Comprehensive implementation of proposed system

As can be seen in the above figure, the proposed system is relatively simple and comprises of very few processes. This further helps ensure that efficient results can be produced, which is of course essential if a user were to use the program in real-time. Breaking down the system, it starts by using a browser extension to retrieve a URL accessed by the client. This extension would essentially work as a proxy, in which all network traffic would parse. Given a link, a client-side web-scraper would then retrieve the key text and image content of a URL using similar methods to as outlined in the data collection phase of this project. Then, using client-side trained instances of both models, the extracted features can be used as input to produce an output in near-real time. If an output were to be labelled malicious, this can then be displayed to a user in the browser through the extension. This system would provide an efficient way to scan websites prior to access, and would not impact user experience when browsing the web.

Given time and resource restrictions, a complete implementation of this system was not developed. Instead, an initial prototype was constructed that demonstrated how the models could be used to produce real-time results. A diagram of the created system can be seen below.

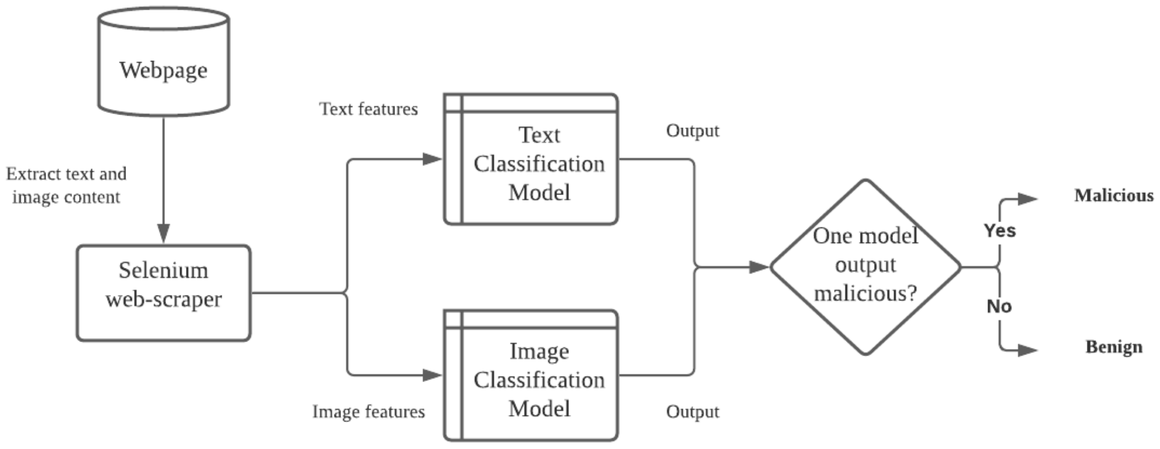
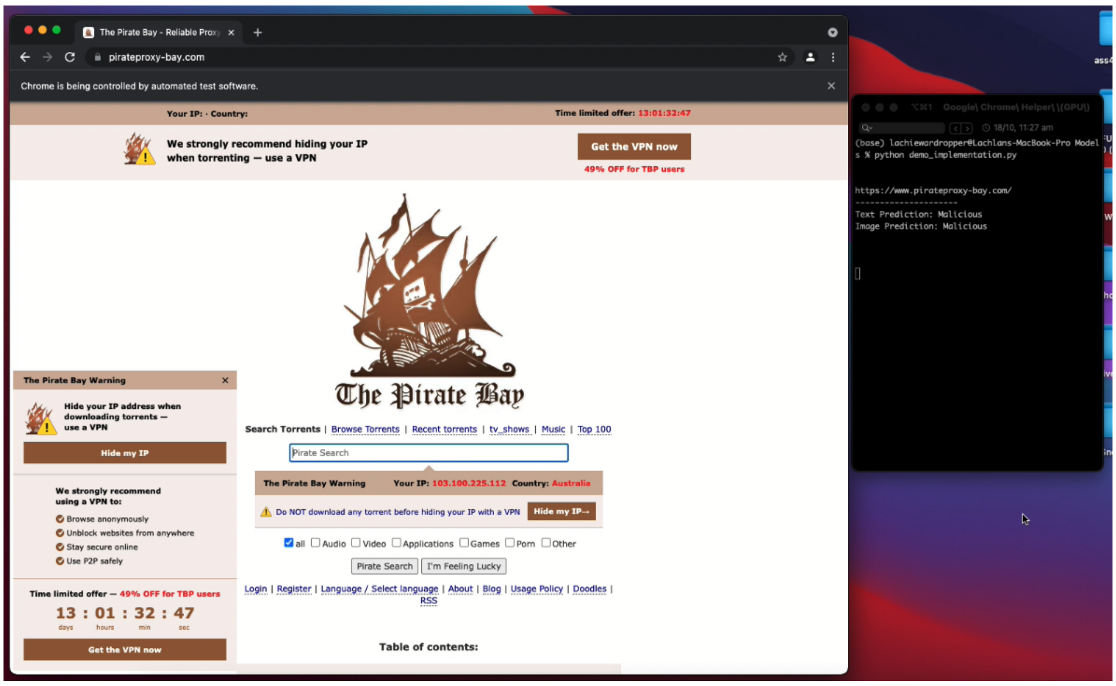

Webpage
Extract text and image content
Selenium web-scraper
Text features
Image features
Text Classification Model
Image Classification Model
Output
Output
One model output malicious?
Yes
No
Malicious
Benign

Figure 4.8: Actual implementation of demo system

In comparison to a complete implementation, this system did not use a browser extension to record user-browser interactions. Instead, the `selenium` library was used to replicate a user browsing the web and clicking links that lead to redirections. For every site visited using `selenium`, the text content would be extracted as well as a screenshot and stored in local storage. This would then be used as input into the already trained classifiers, where an output would be produces in a terminal window. An example screenshot of the implementation is shown below.


The Pirate Bay - Reliable Proxy
pirateproxy-bay.com
Chrome is being controlled by automated test software.
Your IP: · Country:
Time limited offer: 13:01:32:47
We strongly recommend hiding your IP when torrenting — use a VPN
Get the VPN now
49% OFF for TBP users
The Pirate Bay
Search Torrents | Browse Torrents | Recent torrents | tv_shows | Music | Top 100
Pirate Search
The Pirate Bay Warning Your IP: 103.100.225.112 Country: Australia
Do NOT download any torrent before hiding your IP with a VPN
Hide my IP→
all Audio Video Applications Games Porn Other
Pirate Search
I'm Feeling Lucky
Login | Register | Language / Select language | About | Blog | Usage Policy | Doodles | RSS
Table of contents:
The Pirate Bay Warning
Hide your IP address when downloading torrents — use a VPN
Hide my IP
We strongly recommend using a VPN to:
Browse anonymously
Unblock websites from anywhere
Stay secure online
Use P2P safely
Time limited offer — 49% OFF for TBP users
13 : 01 : 32 : 47
days hours min sec
Get the VPN now
⌥⌘1 Google\ Chrome\ Helper\ \(GPU\)
18/10, 11:27 am

```
(base) lachiewardropper@Lachlans-MacBook-Pro Model
s % python demo_implementation.py


https://www.pirateproxy-bay.com/
-----------------------
Text Prediction: Malicious
Image Prediction: Malicious
```

Figure 4.9: Example output of demo system

The terminal window in the above figure displays the labels produced independently by an SVM and CNN model. This is purely designed to show when discrepancies may occur between models. In a working implementation, a single output would be produced based on the following conditions:

Text Model Result	Image Model Result	Output
Malicious	Malicious	**Malicious**
Malicious	Benign	**Malicious**
Benign	Malicious	**Malicious**
Benign	Benign	**Benign**

Table 4.4: Rules for determining labelled outputs of proposed system

This ensures that the total number of true positives is maximised at the cost of a higher false positive rate. Additionally, the prototype produced output labels in close to real-time. The only noticeable inefficiencies were caused by a slow network when retrieving page content. Thus, the implementation created as a part of this project successfully demonstrates the potential for a usable product to be built based off the research conducted.

Chapter 5

Results

5.1 Text Classification

The results of the text classification models show that using text content of a webpage is a viable way to classify a malicious redirection. Of the three models tested, the SVM model had the highest overall accuracy of 98.4%. However, the KNN model was found to have the highest accuracy when classifying malicious labels. A chart comparing the true positive and negative accuracy for the three models can be seen below.

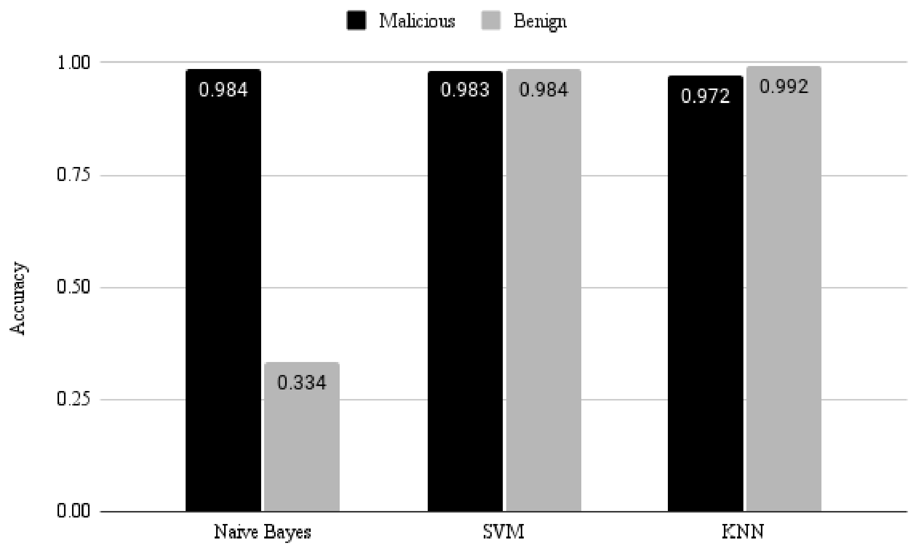


Figure 5.1: Comparison of text classifier accuracy

In addition to accuracy, the models also proved that text classification models have the capability to label websites in real-time. In particular, the naive Bayes model was noticeably more efficient when compared to the SVM and KNN models. However, the KNN model also classified unknown data in negligible time, with each sample taking roughly 1ms to be labelled by the model. Additionally, whilst the SVM model classified the test sample at a seemingly higher rate, the times to classify a single website were still under 3ms. A comparison of the training and validation times for all three models can be seen in the below table.

	Train	**Test**
Naive Bayes	0.0312	0.007
SVM	23.382	9.051
KNN	12.452	2.873

Table 5.1: Train and test times in seconds for text classifiers

Thus, based on the above information, it is evident that the project proved that text content of a webpage can be used to correctly classify it's nature. The evaluation of each model will be analysed in more depth to address potential shortcomings as well as to determine the most appropriate text classifier to use in a prototype implementation.

5.1.1 Naive Bayes

Despite the efficiency of the naive Bayes model, the high false-classification rates suggest that it was not suitable given the dataset. The overall accuracy of the naive Bayes model was 86.82%. However, the accuracy of malicious website classification was only 69.47%. This implies that key text features were not extracted for malicious sites in the training sample. The poor classification of malicious labels is highlighted in the confusion matrix below.

	Predicted Benign	**Predicted Malicious**
Actually Benign	2088	20
Actually Malicious	380	547

Table 5.2: Confusion matrix for naive Bayes model

A key reason for the high false-negative rates for this model is due to the deterministic nature of the naive Bayes algorithm. If a feature set belonging to a validation sample is not seen in the training partition, then the assumed probability is zero. For large datasets with a low number of features this is typically not problematic.

However, this dataset contains an array of unique malicious sites with text features that are unlikely to be shared amongst samples. Furthermore, given there were only 2,239 malicious sites in the training sample, it is likely that the model was not able to build a detailed array of probabilities based on the feature set. Additionally, if the dataset were to be significantly larger and have an even distribution between labels the results would be much better. With this in mind, the results of the naive Bayes model are more of a reflection of dataset limitations than they are model limitations.

5.1.2 Support Vector Machine

The results of the SVM model highlight its ability to classify sparse text data. The accuracy of the model after parameter tuning was 98.8% for both malicious and benign classifications. In order to determine the optimal set of parameters for the model, cross-fold validation was used. It was found that a linear kernel type significantly outperformed a polynomial or sigmoid kernel. The accuracy of the SVM model using a polynomial kernel was 69.4% despite tuning other parameters. This implies the data is linearly separable, and a non-linear kernel is unable to correctly form decision boundaries around the feature space.

Using a linear kernel type, the `C` value was also altered to determine the optimal value. A comparison of `C` values increasing on a logarithmic scale can be seen below:

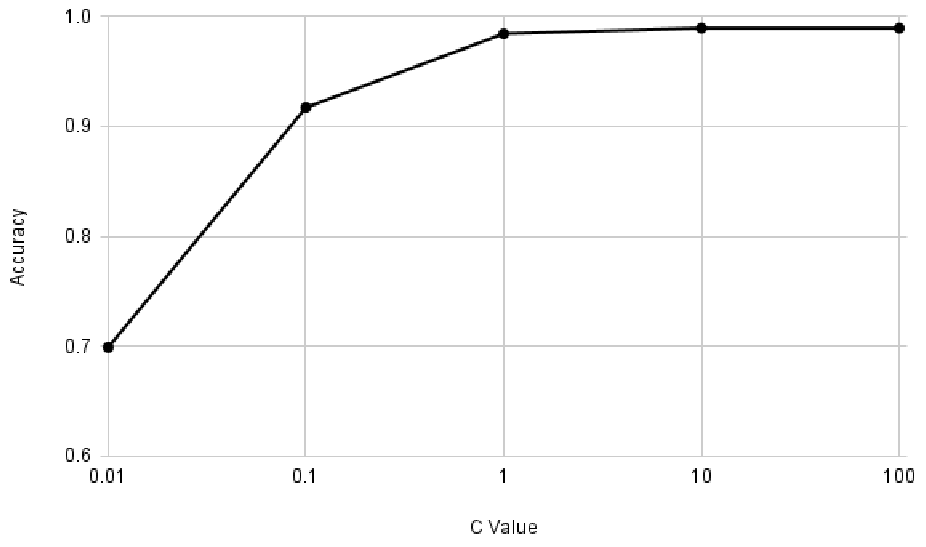


Figure 5.2: Accuracy of SVM model based on C value

As can be seen, the optimal value of C occurs at C = 100 and then plateaus. The value of C determines the margin between the hyper-plane and classes. A low C value will maximise the margin whilst a large C value will minimise this distance. Tuning the C parameter is a trade-off between having the largest possible margin and correctly separating as many instances as possible. Thus, a large C value may be advantageous when used on a small dataset, however when applied to a new set of data over-fitting can likely effect model accuracy. Thus, whilst the maximum accuracy achieved with a high C value is 98.8%, it is likely that a model with a C value of 1 would perform better in a real scenario. As a result, this parameter value was selected despite the 0.4% difference in classification accuracy. Using this value of C, the confusion matrix of the SVM model was as follows:

	Predicted Benign	**Predicted Malicious**
Actually Benign	2075	33
Actually Malicious	15	912

Table 5.3: Confusion matrix for SVM model

Compared to the naive Bayes model, it is evident that the false negative rate is significantly improved. The non-deterministic nature of the algorithm supports a smaller sample size without assuming zero probability for new features. As a result, the model responded much better to the dataset, and would also likely perform better on unknown sources from the web. As a result, this particular model is very viable for detecting malicious website redirections beyond the scope of this project.

5.1.3 K-Nearest Neighbours

Similar to the SVM model, the k-nearest neighbours (KNN) model had a high overall classification accuracy. The model classified the validation data with 97.92% and the training data with 98.84%. In order to determine the optimal value of `k`, the model was trained on all values between 1 and 20, where the best result was then selected. The training and validation accuracy for each value of `k` are shown in the figure below.

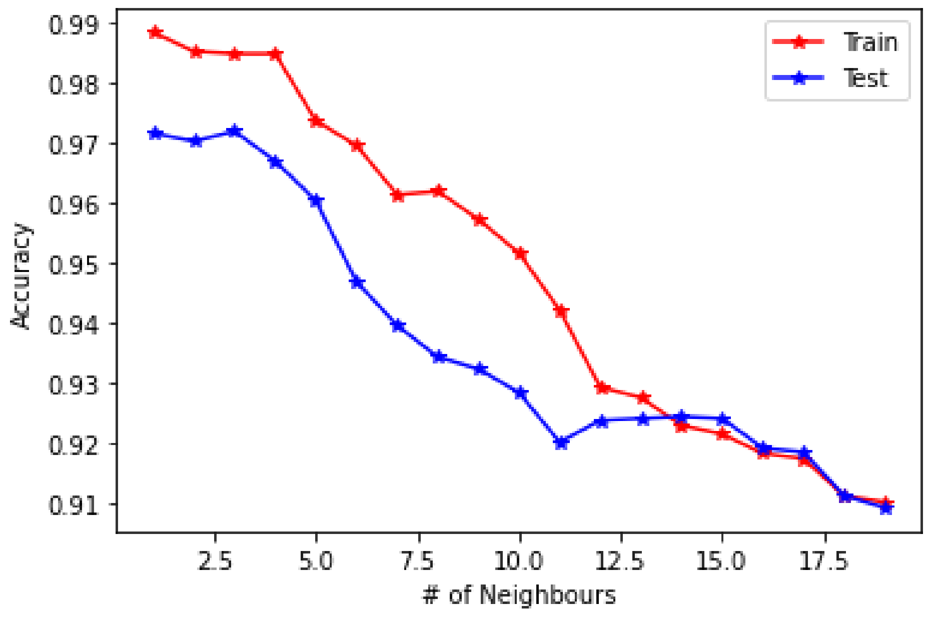


Figure 5.3: Comparison of text classifier accuracy

The comparison between test and training accuracy for varying `k` values show that outliers in the validation set are incorrectly classified by the KNN model. A substantial discrepancy between the accuracy of both sets implies that the boundaries between classes are over-fit to the training data. Furthermore, a low `k` value means that over-fitting is likely to occur given the boundaries are so tight between classes. This is further explicated in the above figure, since the train and test accuracies do begin to overlap when `k` = 14. Despite the decreasing accuracy as `k` increases, if this model were to be used in the wild, it would potentially be less effected by new features with a higher value of `k`. Thus, although the highest accuracy is achieved when `k` = 1, this does not necessarily reflect the optimal value for use on a larger set of webpage data. Having said this, based on this text dataset, the results are still strong. The confusion matrix for the model on the validation set when `k` = 1 was as follows:

	Predicted Benign	**Predicted Malicious**
Actually Benign	2057	51
Actually Malicious	35	892

Table 5.4: Confusion matrix for KNN model

The above matrix outlines the high true positive and negative rates of the KNN model. Based on the dataset, it is evident that this model addresses the goals set out by the project. However, as mentioned, given the value of `k` is 1, the introduction of any new features into the test data would have amplified impacts on model performance.

5.2 Image Classification

5.2.1 Convolutional Neural Network

The results of the convolutional neural network (CNN) model support the hypothesis that it would effectively classify a websites nature based on image content. The overall classification accuracy of the CNN model was 98.92%. Furthermore, looking at the true positive rate of malicious classifications, it is clear that the model was not significantly impacted by the smaller sample size of data. The final true positive rate of the model was 98.8%. An overview of the confusion matrix is outlined below.

	Predicted Benign	**Predicted Malicious**
Actually Benign	2085	23
Actually Malicious	11	916

Table 5.5: Confusion matrix for CNN model

In order to better understand why certain malicious web pages had been incorrectly classified, a visual analysis of the features was conducted. Based on these observations, it was evident that websites wrongly labelled by the model typically contained features and layouts consistent with benign websites. For example, a handful of the wrongly classified malicious sites contained images of people and embedded videos similar to a regular benign website. This represents more of a dataset limitation as opposed to a limitation of the model. Furthermore, this is confirmed based on the observations made of correctly classified malicious sites. When comparing the training and validation datasets, it was clear that the correctly labelled screenshots were all similar in nature to the majority of the training data. In particular, when only looking at malicious torrent and streaming sites, the model had 100% classification accuracy. This reinforces the argument that the imperfect model results were caused by the dataset, as it shows that given enough quantity the model is able to distinguish a significant set of features. Additionally, it was found that torrent and streaming sites generally shared very similar visual features, which also benefited the accuracy of the model.

Based on a comparison between the training and validation results, it is clear that a negligible amount of over-fitting has occurred. Looking at **Figure 5.4**, the training and validation loss values are close to convergence at the lowest point on the graph. This value occurs after the eighth epoch cycle when the training loss is 3.73% and the validation loss is 4.16%. When the validation loss is higher than the training loss, it implies that over-fitting occurred to some extent. As the loss values get closer to convergence the amount of over-fitting reduces. Thus, given the small difference between values as well as the low validation loss percentage, it is evident that the model has performed well with the dataset. The below figure shows how the loss and accuracy values improved as the number of epochs increased.

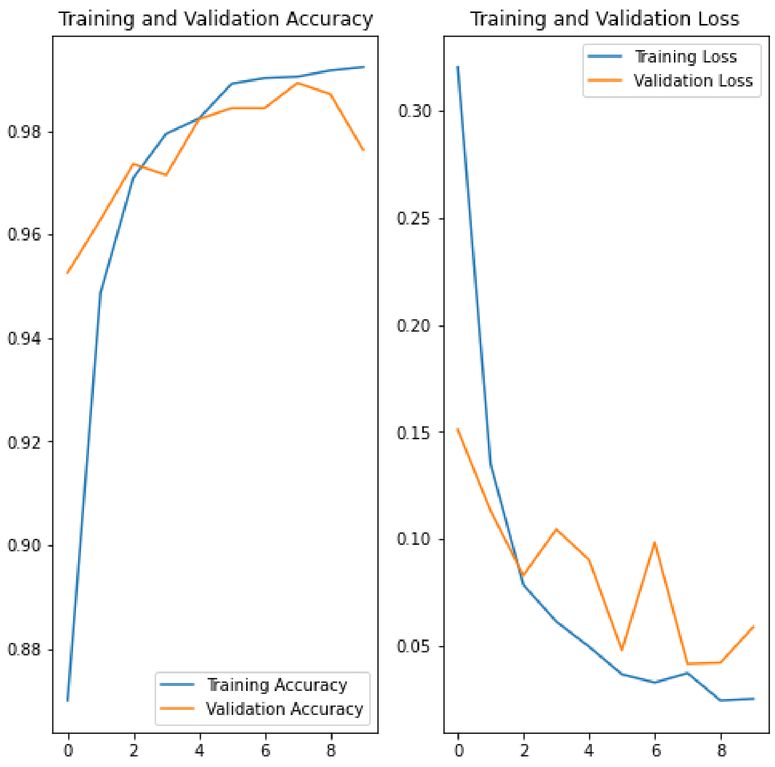


Figure 5.4: Epoch accuracy and loss values for CNN model

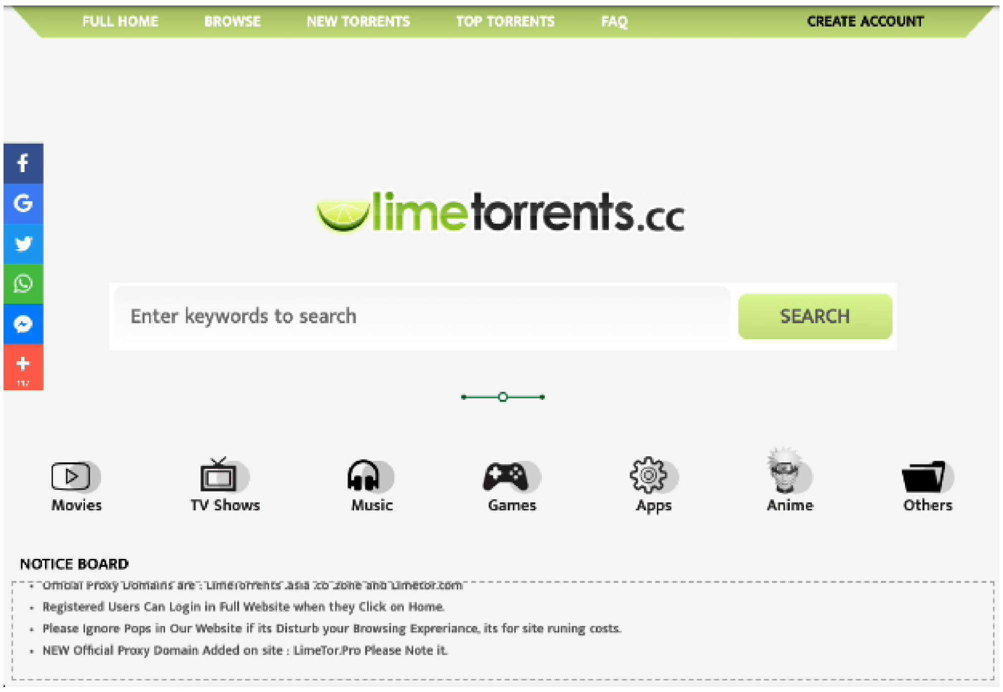
Similar to the loss results, the training and validation accuracies converge after the eighth epoch at a peak of 99.04% and 98.92% respectively. These results exceed any of the text-classification models and highlight the suitability of deep-learning models for this problem. Additionally, despite the long training times per epoch, the validation data was processed and classified in a short amount of time. Thus, the use of a CNN model to label malicious websites was successful, and validates the theory that real-time classifications can be made based on a websites image content and layout.

5.3 Prototype Testing and Results

When developing the final prototype for this project, the most suitable models had to be selected based on a number of results. In particular, the accuracy and the ability for a model to classify sites beyond the scope of this dataset was considered. Comparing the three text-based classifiers, it was clear that the prototype would incorporate either an SVM of KNN model. The decision was made to utilise an SVM model for text-based features given the amount of over-fitting that occurred in the KNN model. Thus, despite the overall accuracies being similar, the SVM model was more likely to better classify unseen data given the decision boundaries did not over-fit to the training data to the extent of the KNN model. Additionally, the CNN model was used to classify website screenshots, as it showed strong results when applied to the dataset.

Two separate testing methods were used to assess the results of the prototype implementation. First, the validation set was tested followed by a small sample of unseen random web pages. These pages were collected via the same process used throughout the project, however with a different set of initial keywords. As a result, the nature of malicious and benign sites differed from those used to train the models.

Testing the implementation against the validation set showed an improved accuracy in comparison to the individual results of both models. The total final accuracy was 99.14% and 99.02% for malicious website classifications. The reason for the improvement was due to the little overlap in misclassifications between models. As a result, using the criteria specified in **Section 4.5**, when one model incorrectly classified a site, it was highly likely the other was correct in its predictions. The main cause for the discrepancies were due to either the text or image contents not aligning with the rest of the dataset. However, it was very uncommon that both the text and image layouts of a website looked benign when it was in-fact malicious. An example of a webpage that produced different labels is shown below.



NOTICE BOARD

- Official Proxy Domains are : Limetorrents.asia .co .zone and Limetor.com
- Registered Users Can Login in Full Website when they Click on Home.
- Please Ignore Pops in Our Website if its Disturb your Browsing Expreriance, its for site runing costs.
- NEW Official Proxy Domain Added on site : LimeTor.Pro Please Note it.

Figure 5.5: Example of webpage producing different model outcomes

In this example, the text-based model correctly output a malicious label whilst the image classifier did not. This was due to the website layout aligning closer to legitimate streaming sites than other torrent pages. However, the SVM model was able to relate words such as torrent, VPN and proxy to malicious sites in the training dataset. This example is representative of many examples that occurred on the validation dataset. It shows that the use of a hybrid model improves the results when compared to both models in isolation. Furthermore, it acts as an additional layer of security against evasion techniques, as malicious actors would have to modify both text and image contents. This is very difficult to do, as it would remove features such as buttons and download links that would otherwise be detected by this solution. A confusion matrix of the results from the validation set is as follows:

	Predicted Benign	**Predicted Malicious**
Actually Benign	2097	11
Actually Malicious	10	918

Table 5.6: Confusion matrix for hybrid model on validation dataset

As can be seen, there was a fairly even spread of false positives and negatives when a hybrid model was used. Ideally, the number of false negatives would be minimal given these are the most dangerous misclassifications in the context of the project. These results could be improved by expanding the dataset to include more edge cases that are currently being labelled incorrectly.

In order to further test the validity of the proposed solution, a separate dataset containing 465 total websites was used. Of this, 157 were labelled as malicious and 308 were benign. The nature of these websites was different to the initial dataset as the majority of malicious sites contained phishing links and e-commerce malware. This data acted as a test to see how the models would react to a set of features that were not a core part of the training dataset. As can be seen by the following result matrix, the models performed surprisingly well on the new set of data.

	Predicted Benign	**Predicted Malicious**
Actually Benign	295	13
Actually Malicious	27	130

Table 5.7: Confusion matrix for hybrid model on unseen data

When looking at the results from the new dataset, it was interesting to see such a high accuracy for malicious website classification. Whilst the accuracy was significantly lower than when the validation data was used, it shows that the models were able to react well to changes in website features. This is indicative of a successful implementation that is does not over-fit to the training data. Thus, if the training partition were to be expanded, it is highly likely the accuracy of classifying new website features would increase.

Another metric that was observed when testing the final implementation was efficiency. The existing prototype is not as fast as it would need to be in order for users to have a seamless experience. It was found that the current text and image extraction method using the `selenium` library was at time causing classifications to take over a second. Based on a sample size of 50 random websites, the average time to extract features and label a website was 0.836 seconds. If this implementation were to be used as a browser extension, this time would likely impact the experience for a user. Given the frequency of redirections as well as the computational resources required to validate a URL, the prototype in it's current state would definitely be noticeable for everyday users. Thus, for this product to be truly non-invasive, the way in which `selenium` accesses and stores data would need to be optimised further.

5.4 Limitations

While the produced results are promising, there are some limitations that limit the viability of the solution in real-world scenarios. First, the data collection process significantly impacted the strengths of the trained models. In particular, given the dataset favoured torrent and download websites, any divergence from this had profound impacts on model accuracy. Whilst further testing found that the models were still fairly accurate when predicting different types of malicious sites, the accuracy rate was not high enough for the solution to be effective in a real-world context. A more efficient data collection process would likely address this limitation. Additionally, the use of a semi-supervised learning model could be used in place of the existing implementation. In doing so, the model could continuously train itself and label data at a pace that far exceeds the existing manual process.

Another limiting factor of the results is that it is not completely immune to the IFrame stuffing evasion technique. This is when a malicious actor hides malicious content inside an `iframe` element which is not detected by a web-scraping bot. Despite image classification being used as well, having sole reliance on this model could be problematic. Without the additional use of a text classification model, malicious actors could design a malicious site's layout to align with benign, trusted websites. This is already common amongst phishing sites, and would certainly impact the overall accuracy of the proposed solution. Having a streamlined data collection and labelling process would help address this limitation, as new trends in malicious site layout would be detected at a much faster rate.

Chapter 6

Conclusions and Future Work

As presented, text and image classification models have been used to detect malicious website redirections, thus meeting the goals set out for this research. A repeatable, open-source process has been outlined to allow anyone to build text and image classification models on website data. Furthermore, the limitations of the data collection process have been highlighted to provide sufficient opportunity for future research to improve upon the results of this project.

When used in combination, the accuracy of malicious classifications for an SVM text-based model and CNN image-based model was over 99% on the validation dataset and 92% on a random sample of websites. This indicates that the models were optimally designed for this dataset and are also viable on specific types of unseen websites. Given that a very specific list of keywords was used to collect data, further collection needs to be performed to expand the types of websites used to train the models. As the breadth and width of the dataset grows, it can be expected that the models will begin to produce higher accuracy results when used to replicate typical user interactions on the internet.

In terms of future applications, this work highlights that precise, real-time classifications of malicious website redirections can be made using text and image models. Additionally, the research shows that a client-side solution can be developed to detect and alert users when a malicious redirection has occurred. This significantly expands upon existing research in the field, and addresses many flaws identified with existing systems. For example, current implementations use botnets to detect malicious sites thus leaving users vulnerable to newly created malicious domains. The proposed method of this project prevents these new forms of evasion techniques from succeeding, and therefore provides improved security for everyday internet users.

The scope of this project focuses on a very niche type of malicious websites and should be expanded to phishing sites as well as malicious pages that do not contain torrents or illegal downloads. This would provide a lot more protection for everyday users, and would help improve overall accuracies of the models. An automated labelling process using machine learning on a range of features would be a suitable next step as it would rapidly expand the size of the dataset. Furthermore, a semi-supervised learning model could be implemented to help label new data and improve the existing models simultaneously. For a solution to be viable for everyday internet users, the training sample size would need to incorporate the many edge cases that exist in user search results. Without such a breadth of webpage types, the number of false negatives would likely have large impacts on the success of a product.

An additional area for future research would be to streamline the feature extraction process. Currently, the use of open-source tools such as `selenium` have significant impacts on the efficiencies of an implementations. This is due to the slow retrieval times of methods given the library is not designed with speed as a priority. For web pages with large amounts of text, these delays can begin to exceed a reasonable length of time for a user to wait. Thus, for an implementation to be used in a non-invasive manner, the speed at which information is extracted should be optimised.

Overall, the project outcomes present a foundation for future research to develop an innovative tool to detect malicious website redirections. The results highlight how flaws in existing solutions can be addressed as well as how modern evasion techniques can be prevented. Further work could optimise the outlined process and provide even deeper insights into how text and image contents of a webpage can be used to detect malicious website redirections.

Appendix A

Source Code

The code used to generate the dataset and models can be found at:

https://github.com/wardsynft/WebsiteRedirectionDetection

The README.md file provides an overview of each script or notebook in the repository.

Appendix B

Bibliography

[1] Netregistry. (Jul. 2019). "Url redirection," [Online]. Available: `https://support.netregistry.com.au/s/article/URL-redirection` (visited on 2021).

[2] K. Lab, "From scared to aware: Digital lives in 2015," Kapersky Lab, Tech. Rep., 2015.

[3] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting frame busting: A study of clickjacking vulnerabilities at popular sites," *IEEE Oakland Web*, vol. 2, no. 6, 2010.

[4] O. Tripp, S. Guarnieri, M. Pistoia, and A. Aravkin, "Aletheia: Improving the usability of static security analysis," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 762–774.

[5] R. Sobers. (Mar. 2021). "34 cybersecurity statistics and trends for 2021," [Online]. Available: `https://www.varonis.com/blog/cybersecurity-statistics/` (visited on 2021).

[6] T. Shibahara, Y. Takata, M. Akiyama, T. Yagi, K. Hato, and M. Murata, "Evasive malicious website detection by leveraging redirection subgraph similarities," *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 3, pp. 430–443, 2019.

[7] S. Yoon, J. Kim, and Y. Jeon, "Analyzing security threats of android-based mobile malware," *Advanced Science and Technology Letters (SecTech 2016)*, vol. 139, pp. 310–315, 2016.

[8] N. Raghava, D. Sahgal, and S. Chandna, "Classification of botnet detection based on botnet architechture," in *2012 International Conference on Communication Systems and Network Technologies*, IEEE, 2012, pp. 569–572.

[9] A. Spadafora, *This cms cyberattack has affected thousands of sites worldwide*, Oct. 2020. [Online]. Available: `https://www.techradar.com/au/news/this-cms-cyberattack-has-affected-thousands-of-sites-worldwide`.

[10] C. Osborne, *Torrent websites infect 12 million users a month with malware*, Dec. 2015. [Online]. Available: `https://www.zdnet.com/article/torrent-websites-infect-12-million-users-a-month-with-malware/`.

[11] L. Chang, H.-C. Hsiao, W. Jeng, T. H.-J. Kim, and W.-H. Lin, "Security implications of redirection trail in popular websites worldwide," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17, Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 1491–1500, ISBN: 9781450349130. DOI: `10.1145/3038912.3052698`. [Online]. Available: `https://doi.org/10.1145/3038912.3052698`.

[12] V. Li, *Why is it so hard to prevent open redirects?* Feb. 2021. [Online]. Available: `https://sec.okta.com/articles/2021/02/why-it-so-hard-prevent-open-redirects`.

[13] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "Automated replay and failure detection for web applications," in *Proceedings of the 20th IEEE/ACM international conference on automated software engineering*, 2005, pp. 253–262.

[14] C. Inc, Sep. 2020. [Online]. Available: `https://techdocs.broadcom.com/us/en/symantec-security-software/identity-security/siteminder/12-7/configuring/web-agent-configuration/session-protection/redirect-a-user-after-a-session-time-out.html`.

[15] PortSwigger, *Dom-based open redirection.* [Online]. Available: `https://portswigger.net/web-security/dom-based/open-redirection`.

[16] D. Liu and J.-H. Lee, "Cnn based malicious website detection by invalidating multiple web spams," *IEEE Access*, vol. 8, pp. 97 258–97 266, 2020.

[17] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson, "Clickjacking: Attacks and defenses," in *21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 413–428.

[18] *Url redirection - attack and defense*, Jun. 2020. [Online]. Available: `https://www.virtuesecurity.com/kb/url-redirection-attack-and-defense/`.

[19] U. U. Rehman, W. A. Khan, N. A. Saqib, and M. Kaleem, "On detection and prevention of clickjacking attack for osns," in *2013 11th International Conference on Frontiers of Information Technology*, IEEE, 2013, pp. 160–165.

[20] NoScript, *There's a browser safer than firefox... ...it is firefox, with noscript!* Nov. 2014. [Online]. Available: `https://noscript.net/`.

[21] Google, *Noclickjack*, Apr. 2019. [Online]. Available: `https://chrome.google.com/webstore/detail/noclickjack/ljkbghpfokmghicbbbemglhflkmjampc`.

[22] B. Sun, M. Akiyama, T. Yagi, M. Hatada, and T. Mori, "Automating url blacklist generation with similarity search approach," *IEICE TRANSACTIONS on Information and Systems*, vol. 99, no. 4, pp. 873–882, 2016.

[23] Google, Oct. 2021. [Online]. Available: `https://transparencyreport.google.com/safe-browsing/overview`.

[24] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09, Paris, France: Association for Computing Machinery, 2009, pp. 1245–1254, ISBN: 9781605584959. DOI: `10.1145/1557019.1557153`. [Online]. Available: `https://doi.org/10.1145/1557019.1557153`.

[25] H. Choi, B. B. Zhu, and H. Lee, "Detecting malicious web links and identifying their attack types," in *Proceedings of the 2nd USENIX Conference on Web Application Development*, ser. WebApps'11, Portland, OR: USENIX Association, 2011, p. 11.

[26] Archive.org. [Online]. Available: `https://archive.org/web/`.

[27] McAfee, "The web's most dangerous search terms," McAfee, Tech. Rep., 2006.

[28] N. Leontiadis, T. Moore, and N. Christin, "Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade.," in *USENIX Security Symposium*, vol. 11, 2011.

[29] Y. Lindell and E. Waisbard, "Private web search with malicious adversaries," in *International Symposium on Privacy Enhancing Technologies Symposium*, Springer, 2010, pp. 220–235.

[30] A. Aryan, *Google search unofficial api for python with no external dependencies*, Jul. 2021. [Online]. Available: `https://pythonrepo.com/repo/aviaryan-python-gsearch-python-third-party-apis-wrappers`.

[31] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, 2003.

[32] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng, "Some effective techniques for naive bayes text classification," *IEEE transactions on knowledge and data engineering*, vol. 18, no. 11, pp. 1457–1466, 2006.

[33] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.

[34] *Sklearn.svm.svc.* [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`.

[35] I. Goodfellow, Y. Bengio, and A. C. D. Learning, *Cambridge, ma*, 2016.

[36] A. Dertat, *Applied deep learning - part 4: Convolutional neural networks*, Nov. 2017. [Online]. Available: `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`.

[37] D. Sahoo, C. Liu, and S. C. Hoi, "Malicious url detection using machine learning: A survey," *arXiv preprint arXiv:1701.07179*, 2017.