Imperial College London

Individual Project Final Report

Department of Computing

Imperial College of Science, Technology and Medicine

PhishingLine: Hybrid Phishing Classifier with Logo Detection

Author:

Kabeer Vohra

Supervisor:

Dr. Sergio Maffeis

Second Marker:

Dr. Luis Muñoz-González

May 25, 2018

Submitted in partial fulfillment of the requirements for the Computing Masters of Engineering degree of Imperial College London

Abstract

Phishing attacks are one of the most lucrative forms of online fraud today. In these attacks, criminals lure victims into disclosing sensitive information by masquerading as another legitimate entity. Unlike traditional malware, which typically is automatically detected by local anti-virus software, phishing attacks are online and therefore need to be identified and blacklisted. The sheer volume and rapidity with which phishing pages are created, requires that browsers need to update from popular blacklists, such as Google Safe Browsing, every five minutes in order to keep users reasonably well protected.

Recent research has identified machine learning as an effective way to improve automated phishing detection. In this report we present our hybrid classifier, a novel approach which combines analysis of page attributes with cascading classifiers on extracted images and screenshots.

We have developed a platform which performs comparably with the current state of the art, whilst detecting pages with the equivalent of research leading performance, and providing an easily extensible platform for future collaborators. Additionally, our learning platform is designed to be able to automatically adapt to new styles of malicious pages and fit to new phishing techniques.

Acknowledgments

I would like express my gratitude to my supervisor, Dr. Sergio Maffeis, thank you for all your helpful feedback and guidance in our regular meetings as well as being available on slack most hours of the day.

I would also like to thank my second marker, Dr Luis Muñoz-González for assisting me on the machine learning parts of the project and being available for meetings at a moments notice.

To Marco, Clemens, Guillherme, Alessandro and everyone else at Lastline. Thank you for all the help, allowing me to work from your office, regular meetings and providing me with all the data and structure to complete this project.

To Dad, my best mate. Thank you for all you have given me throughout my entire life and all the football and music we have shared.

To Mum. Thank you for all of the guidance and counselling you have given me, our regular nights out together are always special.

To Aneet and Neha. Thank you for all the laughs and fights we have had in equal proportion over the past 19 years.

To Clelia. Thank you for dealing with all my moaning over the past year, there's more to come.

To Gaurav, David, Chinmay, Elyas, Florian, Mahir, Jack, Harry, Nihal and all my other friends over the course of this degree. Thank you for all your help and all the good times we shared.

Contents

1 Introduction

Social engineering is the art of hacking the human psyche. The victim is tricked into divulging confidential information or to perform specific actions. It is not merely a modern phenomenon, for time immemorial people have used this method to gain access to resources the victim may have. Recently, Michael Wolff 'scooped' the world's journalists when he was able to gain permission to be a fly-on-the-wall inside the White House simply by using basic strategies of social engineering. This allowed him to write one of the first books about the Donald Trump presidency [24].

With the advent of the digital era, and the omnipresence of the internet in today's society, criminals have realised how easy it is to move this practice online and weaponise their efforts. This practice is known as **phishing**. Criminals can use this illicitly collected data to steal money, perform identity fraud or even to make physical attacks using obtained home addresses.

Phishing attacks will often initially masquerade in the form of an email alerting the victim that something is wrong, thereby creating a sense of urgency. This may result in users performing actions, without taking due care that the email is legitimate. Common elements of such emails include logos of target websites, language to create a sense of urgency and a link to the phishing website. An example phishing email can be found in figure 1.1.

Unlike many elements of cyber security which are a constant arms race between researchers and attackers, 90% of phishing attacks examined over the 10 months between late 2016 and mid 2017 were replicas of attacks already detected [12]. Whilst we are able to roll out patches for known security flaws swiftly, the size and nature of the internet makes phishing a much harder problem to tackle.

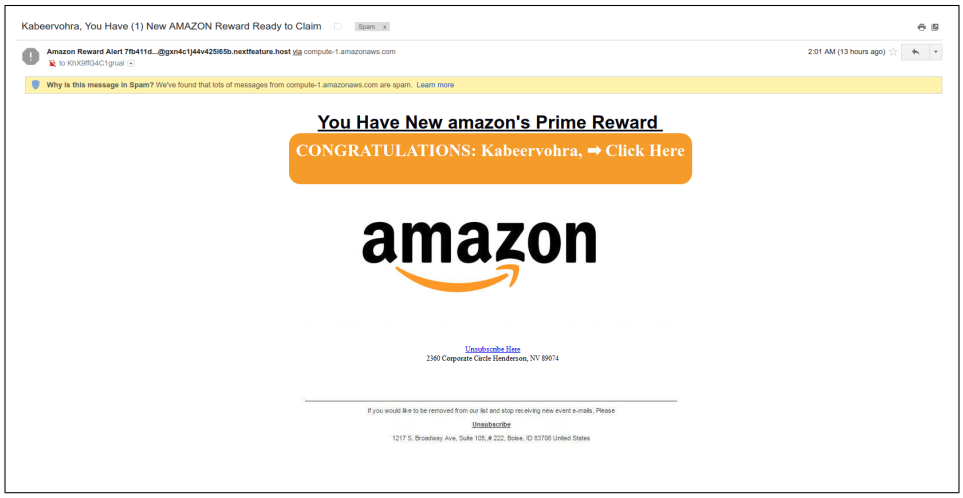


Figure 1.1: Example phishing email received advertising fake prize in order to gain access to Amazon login credentials

1.1 Motivation

Various methods of classification in the literature today have different benefits. Cantina had an extremely high true positive rate but also suffered from many false positives [53]. Whittaker et al. had similar issues with their methods of classifying pages using PageRank [52]. All the phishing pages they investigated had a very low PageRank, however this was true for many legitimate pages as well. For this reason a hybrid approach has been used in more recent techniques, where classifiers can use a combination of the most effective approaches to gain the highest sensitivity and specificity.

Creating a modular solution allows us to implement different methods and combine the results of these independent classifiers. In this way we have the ability to easily extend the classifier by creating further modules and consistently improving the accuracy of the classifier.

Often in phishing attacks, criminals may choose to imitate the brand rather than

simply creating clones of legitimate pages. Whilst these pages may not have any visual similarity with pages on the target domain, they will need to include the branding of the target company. Without this branding, it would be difficult to convince the victim that they are visiting a page originating from the target brand. By detecting this on the page, we would then be able to check if this page actually resides on the domain of the brand, or elsewhere. We also know that, according to the anti-phishing working group, in Q4 of 2017 there were a total of 640 unique brands targeted and a few hundred companies attacked regularly [32]. For this reason it should be feasible to be able to gather the data for all of these targets using a script and then use this data to try and identify the target brands.

Phishing pages today are mostly created using phishing kits. These kits allow the attacker to quickly generate fully-fledged phishing pages, ready to be deployed. According to Marco Cova et al., newly detected phishing kits are often derivatives of previous kits that are still currently active [11]. Therefore it is self-evident that the HTML provided by these kits will be similar to previously detected kits and the same will be true of the phishing pages which use these kits. From this assumption we can try and compare the DOM trees pages which we wish to classify with known phishing pages as a metric to evaluate if we believe the page is using a derivative of the kits which we have discovered.

In order to compare DOM trees, it is interesting to try and compute bi-simulations between them. To do this, we can take pairings between the two DOM trees, and then use a fitness function to compute which pairs we wish to keep, and which we want to ignore. By applying a progressive monotonic function, we are able to reduce the obtained set, and then use this data to compute a metric on the similarity of the two pages.

1.2 Contributions

In this report we developed a hybrid machine learning based approach to phishing classification. We will summarise the main modules of our final pipeline here.

- **URL (Section 6.4.3):** We analyse the URL for many of the common features that attackers may use in order to trick the victim into trusting the page. The URL provided some of the most important features in our overall classifier and can be useful to use as a single module if we wanted to prioritise performance only.

- **Word Frequency (Section 6.4.4):** Our word frequency module allows us to analyse the direct textual content on the pages for the highest frequency. Often, phishing websites use similar terminology and therefore we can automatically learn these words as the pages evolve.

- **HTML (Section 6.4.6):** In this module we extract features in order to profile the structure of the page to feed into our classifier. We attempted to compute the DOM similarity between phishing pages but found it difficult to do effectively with the structure of our classifier.

- **Page Resources (Section 6.4.7):** We analysed the locations of the resources that the page loaded in as well as where the page sent data. Often phishing pages need to load resources from other hosting sources or directly from the page which they are trying to imitate. They will also need to exfiltrate the stolen credentials to their servers which may be different to the ones on which the page is hosted.

- **Logo Detection (Sections 6.4.8 and 6.4.9):** When trying to determine the brand of a page, phishing websites use familiar logos to trick victims visiting the page. We use various computer vision techniques to detect the logo on the page and then subsequently determine whether the page originates from the source corresponding to the logo.

2 Evaluation Techniques

In the following chapters of this report we will present our classifier and other approaches in the literature which have been evaluated using some of the techniques below.

2.1 Sensitivity and Specificity

Classification algorithms are measured by their *sensitivity* and *specificity*.

The sensitivity of a test is judged by its *true positive rate*. This is the proportion of websites which are phishing sites and have been classified as such. A highly sensitive test is one that classifies most of the phishing sites correctly. An example of which is Cantina developed by Y. Zhang et al [53].

With a highly sensitive test we can be quite certain that if the test classified the webpage as benign (non-phishing), then it would be a correct judgement. We would not be confident, however, that if the test marked a page as phishing, whether it was correct or if the page was a benign page.

The specificity of a test is judged by its true negative rate. This is simply the legitimate websites that have not been classed as phishing. If we have a high true positive and high true negative rate, then we would have a good classifier.

Conversely, false positive and false negative results are where the test incorrectly classifies the data as a phishing site or a non-phishing site respectively.

2.2 Confusion Matrix

A confusion matrix is a table which allows us to classify our results into true positive, true negative, false positive and false negative.

		True classification	
		Non-phishing	Phishing
Test result	Non-phishing	TN	FP
	Phishing	FN	TP

Table 2.1: An example confusion matrix

2.3 Complementary Cumulative Distributions

Complementary Cumulative Distributions (CCDF) are used in order to show what percentage of the data set is above a certain threshold. An example can be seen in figure 3.1. Here we can see that there are no phishing pages with PageRank above 0.5 since the graph intersects with the x-axis at this point.

2.4 Binary and Multiclass

When classifying a page we can use either a binary or a multiclass classifier:

Binary classifiers will output a single result as to whether the page is phishing or non-phishing.

Multiclass classifiers can provide a set of output metrics which can then allow us to assess whether the page is phishing or not.

2.5 ROC Curves

ROC curves are common tools in binary classification. They allow us to evaluate what true positive rate we can achieve when considering the highest false positive rate that we can accept. Analysis of the ROC curves provide us with information to select optimal thresholds for the problem we have, dependent on whether we care, more about false positives or false negatives. Lines on the ROC curve that fall below the dotted line are considered to be worse than simply selecting a random classification with a success rate of 50%.

2.6 Cross-Validation

Overfitting is a common problem within machine learning techniques. It is where the classifier can start learning the selection bias which we have within our data set which will not help us to classify new examples very well. Generalizability is important in trained models and overfitting kills this. Our initial testing may seem positive but we need methods to ensure that we are accurately evaluating our classifier so that when it is fed new data it is still able to classify correctly.

To solve this problem we use k-fold cross validation, this is a technique used to split the training data into k separate splits and then use $k-1$ splits to train and one to test, we repeat this k times with the single test subset rotating among the k subsets. This allows us to ensure that the entire test set can be correctly classified and that our classifier is not over fit to a single test set.

3 Current State of Phishing

3.1 Primarily Affected Industries

According to the anti-phishing working group, the top 3 targets for phishing attacks in the second half of 2017 were payment, financial institutions and webmail [32].

3.1.1 Payment

The payment industry makes up 42% of all phishing attacks. Easy access to victim's money is the obvious driver for this high metric. Typically smaller payment services have less sophisticated anti-fraud departments than do the banking industry, making them easier targets for potential attacks.

3.1.2 Financial

Whilst these companies typically have sophisticated anti-fraud protections in place, financial institutions still make up 15% of all attacks, criminals realise that the most lucrative method is still to go directly to the banks. With the rise of Bitcoin and other anonymised decentralised currency systems, it is becoming much harder for institutions to protect users' money and reverse fraudulent transactions [4].

Online banking services often implement phishing prevention best practices in order to warn the user if they are not on the legitimate site. Santander for example implement the recognised image technique where they assign an image to the user to be displayed before they login. If the user reaches a login page without the image this should hopefully arouse suspicion.

3.1.3 Webmail

Exploiting a user's webmail account gives access to all online login details associated with that email address. Due to this expanded benefit, fraudulent webmail access is highly valuable and makes up 16% of the overall market.

3.2 Attack Characteristics

3.2.1 Lifetime

Phishing attacks are generally alive for a very short period of time before they are rendered useless. The lifespan of an attack is measured as the time between the first and last observed attack instance.

In the research conducted by Qian Cui et al. 34% of phishing attacks only last between 1-7 days after they were made active, longer attack classes can last between 2 and 10 months [12]. The average time for an attack class to be active is 25 days.

Although these attacks are closed down so quickly, they clearly still remain extremely lucrative. Attackers generally set many attacks at once and profit during the short window of opportunity that the phishing attacks are active.

3.2.2 Phishing Kits

Phishing kits are very popular in the phishing world today, they allow would-be criminals to create phishing exploits with little technical skill. These kits are usually sold anonymously on the dark web by malicious developers through the use of Bitcoin and other untraceable cryptocurrencies [4].

The kits vary in technical complexity, they can be very basic pages which mimic the

brand and have no credential verification[1]; or they can be more complex attacks which are virtually indistinguishable from the legitimate page and perform validation against a real API to ensure the credentials are valid.

Analysis of a more sophisticated kit by ProofPoint, aimed at PayPal, showed that form validation was done only on the username field by using a deprecated API provided by PayPal themselves [23].

After the user proceeds to login, the kit also asks the user to confirm bank credentials to secure their account, all this information is logged in the backend portal of the phishing kit. This particular phishing kit also allows for a 'selfie mode' which allows the attacker to access the webcam of the victim as well.

3.2.3 PHP

Marco Cova et al. used various methods in order to obtain phishing kits that are being used today [11]. Each of the 584 kits obtained were written in PHP. PHP is a popular language because it is widely supported and enabled on most web servers.

The use of PHP was leveraged in the implementation of the PhishEye honeypot [20]. Two Apache HTTP modules `mod_php` and `mod_security` were extended in order to track actions that were taken in the PHP exploits.

From **mod_php** they modified three handlers, `rfc1867_post_handler`, `php_std_post_handler` and `php_default_treat_data`. These three handlers deal with any form posts, POST and GET requests. All of these were recorded along with the IP address of the caller.

mod_security was used in order to perform data mangling so that the criminal would not receive any sensitive information.

[1] Credential verification is when the phishing website ensures the input details are valid against the legitimate website before letting the user proceed

3.2.4 URL Obfuscation

For attacks which are not hosted on the target page, attackers generally use four different techniques in order to obfuscate the illegitimacy of the URL from the user [11].

Technique I is to purchase commonly misspelt domains and host them as phishing websites. With this technique the URL seems legitimate, since it is virtually identical to the target URL. Apart from the redirection benefits, say from an email, the hacker is also able to serendipitously capture victims who have simply mistyped the URL themselves. An example of this could be `https://gooogle.com`. Whilst large organisations usually purchase most of the commonly misspelt URLs, this can be an effective method for second tier company websites.

Technique II is to purchase a short domain and have the full correct URL prepended to this short name. An example of this could be `https://bank.barclays.com.vit.ly`. This methodology is the most effective if the target domain is secure and there are no obviously misspelt domains available.

Technique III is when the attacker compromises another domain which has a backdoor and hosts the attack on there. The backdoor URL will contain the name of the target of the attack, for example `https://compromiseddomain.com/barclays`. This method provides a lower overhead for the attacker since they do not need to register a separate domain themselves. It also reduces the risk they take of being caught from information related to the registration of the domain they use for the attack.

Technique IV is when the domain has absolutely no relation to the target of the attack. These are the least effective methods but also have the lowest overhead. For these attacks, the criminal needs only to find a domain which has a backdoor and uses this to host their attacks.

It is important to note that using HTTP Secure protocol (HTTPS) is a very effec-

tive tactic that is used. By displaying the secure lock in the browser due to a valid SSL certificate, attackers are able to maintain a level of trust without deep inspection of the URL itself by the victim [33].

3.2.5 Data Exfiltration

During phishing attacks, data exfiltration is a key issue for criminals. Either they can have the data written to a log and stored on the server which they can then retrieve or they need to have the data sent over email to an account that they use to collect the results.

Email exfiltration is easier to detect by monitoring the outgoing SMTP and also contains the email address of the attacker. It is, however, preferred by some attackers because it avoids the need to have to re-access the compromised server to retrieve the data. This server may be patched or being monitored if the phishing kit has been detected.

3.2.6 PageRank

PageRank is the algorithm that Google uses in order to determine which websites will be prioritised in their search results. It is a constantly evolving system which continually improves the quality of the results, and counteracts organisations which attempt to artificially promote their websites to the top.

The algorithm is based, amongst other variables, upon the number of quality links from other reputable sources a website may have; in order to determine its importance as a website and provide legitimacy. Due to the short typical lifespan of an attack, the return on investment associated with improving PageRank for a specific malicious page is too low to justify the efforts required.

As seen in Figure 3.1 by Colin Whittaker et al. none of the phishing pages have a high PageRank attribute[52]. This is a good classifier for the detection of phishing

pages but, by itself, will also contain many false positives, due to the large quantity of legitimate pages which also have a low PageRank.

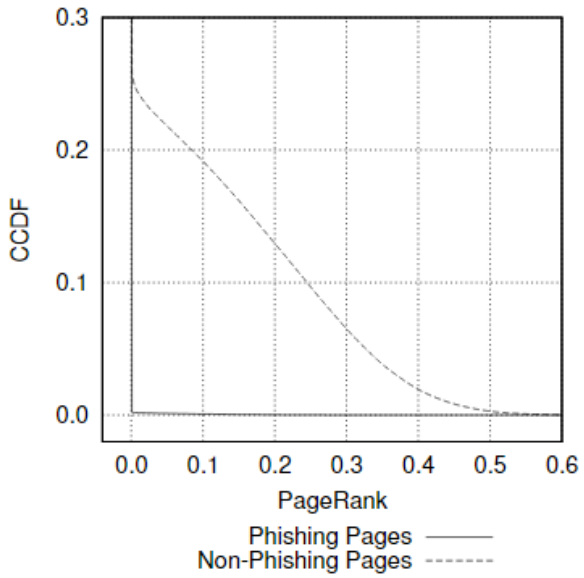


Figure 3.1: CCDF of PageRank for phishing and non-phishing pages [52]

3.3 Defence Mechanisms

3.3.1 Browser Extensions

A study conducted by Y. Zhang et al. from Carnegie Mellon University evaluated many different anti-phishing toolbars [54]. These approaches all used a level of pre-processing before the page loaded and then displayed the level of risk associated with each page visited in a toolbar on the browser. Many methods were employed by these extensions such as investigating the source of the hosting with each page, using heuristics on the layouts of the pages, and comparing them against blacklist data.

3.3.2 Blacklists

Blacklists are lists of URLs which have been flagged as being malicious, these are generally inbuilt into the browser and the legitimacy of pages are checked before they are

visited. With the rise in use of mobile phones, many of their browsers ban the use of extensions, therefore promoting the use of blacklists. In addition, the extra load time overhead of pre-processing the page before display makes inbuilt browser page blacklisting the approach that current researchers use.

There are a wide range of methods that are employed today to improve classification of phishing websites in order to keep the blacklists updated frequently enough. Due to their monopolistic nature, Google is able to keep updated crawls of new pages across the internet to refresh their Safe Browsing lists frequently, allowing them to incorporate this within their widely used search engine and browser [42].

3.3.3 Reporting

PhishTank is the industry standard website where all phishing attempts are reported and confirmed [35]. They provide an open API for anyone to access their data. Submitted URLs are verified and, if incorrect, removed very quickly by a number of researchers in the field.

3.3.4 URL Checking

As discussed in section 3.2.4, there are many characteristics of URLs that phishers use in order to mask them from the user, these give us the opportunity to employ various techniques to detect suspicious addresses. Common features of phishing URLs that can be automatically detected are:

IP address: if the URL detected uses an IP address for it's hostname, it is more likely to be a phishing page.

Host components: if the URL has many components then it is likely to fall under a page which has used Technique II. Using the example in section 3.2.4, `https://bank.barclays.com.vit.ly` has 5 components whereas the legitimate website `https://bank.barclays.com` only has 3.

Individual features: the URL may contain common trademarks of the official company, for example including 'abbeynational' for an attack against the Santander group.

3.3.5 Honeypots

Honeypots are commonly used in cyber-security to obtain more information about the characteristics of an attack. They are websites which contain deliberate vulnerabilities and allow researchers to monitor their activity.

PhishEye was a honeypot developed by Xiao Han et al. [20]. The honeypot was hosted using Amazon EC2 which is a popular target for attackers and contained a selection of vulnerable PHP files.

A number of methods were used to nullify the potential damage caused by this honeypot. Client side data mangling using JavaScript, and server side data randomisation, was used to ensure that any data that the honeypot would collect from victims was scrambled before the attacker attempted to exfiltrate it.

Data entry and exfiltration were monitored throughout the lifetime of the phishing attacks. Since a common method of exfiltration is via email, SMTP channels were monitored. Although this did not stop any live phishing attacks, it allowed the researchers to study and record the methods used and helped identify attackers.

3.3.6 Hosting

There are two different types of phishing attacks [11]. Hosted attacks are where the phisher compromises a legitimate server and installs a phishing kit. Self-hosted attacks are where the phishing website is hosted on a domain purchased by the attacker.

Around 1500 attacks each month are performed using free hosting websites, the most popular being `000webhost.com`, `myjino.com` and `freeavailabledomains.com`. Ama-

zon EC2 websites are commonly targeted due to being self hosted and managed by users rather than having professional managed solutions in place. Professionally managed solutions will often have better built-in security than user managed ones.

The classic configuration of a domain is to purchase the website and email hosting from the same provider. When these domains have been compromised the attacker will typically change one of the DNS records so that this assertion is no longer true.

Hosting an address on the A, MX or NS record of a website is considered suspicious. Since hosting usually requires tier 1 speeds; if we detect these speeds on one of the records, it is a key signal that the website has been compromised.

4 Proposal

When a malicious entity creates a phishing page, they usually leave many digital signals which help us to identify whether the page is phishing or not. Machine learning techniques have proved an incredibly effective way of detecting spam emails to the point where spam detection has become a common way to teach people the elementary methods in machine learning [6]. It seems natural to use this approach in phishing detection since they are problems of a similar nature.

Whether an attacker deliberately wishes to fool users by embedding information into the URL in order to make it look legitimate, or if they have compromised vulnerable websites in order to host their attacks, there is plenty of information a classifier can gain, with low computational requirements, by initially analysing the URL. As mentioned in the objectives of the project, the proposed system is performance conscious, for this reason we have to make some trade-offs in the implementation.

When attackers create phishing pages, the most common approach is to use off-the-shelf *phishing kits*. These kits generate complete HTML pages, including forms, images and other included resources. Often the attacker needs to simply obtain these kits, find a page with a security weakness to exploit, and upload the kit in order to have a fully working phishing page. In addition to this, new phishing kits are often derivatives of old phishing kits, adding new features but with largely similar HTML structures. For this reason we will attempt to try and detect structural similarities between different phishing pages.

Due to the ease of creating clones and the added familiarity to a victim, phishing pages generally are imitations of real pages. For this reason, many implementations of phishing detection involved comparing the page against the legitimate page as S.

Haruta et al did in their visual-similarity based phishing detection [21]. In theory, this is a highly effective way of detecting whether the page is attempting to copy another page, however, in practice, we are usually trying to classify individual pages without further context. This means that without the knowledge of which brand the page is trying to imitate, we would have to compare the page to a large number of brand pages which is both computationally expensive, and non-exhaustive, since we would need to compare against a large number of pages imitated per brand. Due to the above, we will not attempt to classify phishing pages in this way in our methodology.

While it is difficult to match a phishing page against its corresponding legitimate page, phishing pages need to display the brand image which they are imitating in order to convince a victim that they are on the legitimate brand's page. To do this they need to display the logo of the target somewhere. This logo can be loaded from the legitimate domain itself or hosted on an image hosting site. Often attackers will also apply some basic level of image manipulation which is undetectable to the human eye but makes it much harder for automatic matching systems to detect the logos. We will attempt to use different methods of image matching to detect logos within the screenshot of the page.

When loading phishing pages on compromised domains, attackers may need to use HTTP redirects to redirect traffic from certain pages to their pages. In addition to this, phishing pages have a short time to live as they are blacklisted or taken down frequently. For this reason, the status code returned by the page is an important heuristic to investigate.

4.1 Deployment

The classifier is intended to be used as part of the analysis performed at Lastline and requires an easy way to query the system whilst also being time-critical. With our analysis, there is a lot of pre-processing that needs to be done before the classifications can be made. For this reason a client-server approach for classifying files is the most

efficient way to return a result without incurring the loading overhead for each page that needs to be classified.

The server will be run on `localhost` and will be queried with a REST interface with a JSON input and a JSON response.

5 Implementation Technologies

5.1 Machine Learning

Machine learning is an old discipline in the area of computing based heavily on statistical principles. Models are created through the process of *training*, whereby using copious amounts of pre-classified data, features that are characteristic about the data can be learnt. They can then use these features to classify new data that they have received.

5.1.1 Feature Extractors

Features are how machine learning models are able to extract statistical patterns from input data into numerical variables, these can then, in turn be fed into the classifier. These variables are typically binary features, however they can also be integer or float values.

5.1.2 Pipelines

When applying various feature extractors to an input, there is a pre-defined set of transformations that we apply when training and evaluating the classifier. All of these transformations can be compiled as part of a data pipeline. When we call `transform` or `fit_transform` on the pipeline, this will apply each component of the pipeline to the input and use the components to produce feature sets relating to the data.

5.1.3 Classifiers

There are a number of different types of classifiers that are suited to different machine learning based problems [16].

5.1.4 Linear Regression

Linear regression is mainly used to classify continuous variables such as cost of houses, numbers of calls, and total sales. The algorithm is based on plotting the various data points on a graph and then computing the line of best fit between them. The line of best fit is then used when providing a new testing sample to predict a new value.

Simple linear regression concerns the use of a single independent variable to fit on whereas multiple linear regression uses multiple linear variables. Linear regression is able to accept discrete data as input but will always have a continuous output distribution.

5.1.5 Logistic Regression

Logistic regression is used to estimate discrete values using a set of independent variables and outputting a binary classification. The classifier is computed in a similar way to linear regression but uses an activation function to make the output binary.

5.1.6 Stochastic Gradient Descent

Stochastic gradient descent methods are used when the weights of a classifier are to be reduced by computing the derivative of the risk and moving against the gradient. It is computed in the following manner:

$$w = w - \eta \Delta Q(w)$$

Where $\Delta Q(w)$ is the error and η is the learning rate of the classifier which we can manually specify.

5.1.7 Decision Tree Classifier

Decision trees are best used when there are sets of features which evaluate to the same class in the training data. The classifier divides the feature set into sub parts and

terminates when it has reached sets where the features belong to a single class. It then computes decision trees along the set of discrete variables within each feature set that has been created.

The impurity is then calculated from the probability of each item in the decision tree as:

$$-\sum p(x) \log p(x)$$

The classifier then attempts to minimise the impurity and maximise the information gain in each iteration.

5.1.8 Random Forest Classifier

Random forest classifiers are sets of decision trees which span different subsets of the feature space. The result of all of the independent decision trees is then aggregated and used for the final classification. The advantage of using random forests over normal decision trees is the reduction in noise by avoiding to combine features which do not correlate well together.

5.1.9 N-Grams

N-grams are feature extractors which allow the classifier to learn and detect commonly used phrases within the text.

5.1.10 Scikit

Scikit-learn is the industry standard machine learning suite for Python which helps to make many of the main machine learning implementations readily available with a common interface.

5.2 Modal Logic

When comparing two trees we can use elements of Modal Logic to try and compute the similarity between them. We can start by converting our trees into the form of Kripke models, once we have done that we can compute the bisimulations between the two trees. [22]

We will initially define the powerset notation $\wp$ as the set of all subsets of its input:

$$\wp(S) = \{U : U \subseteq S\}$$

5.2.1 Kripke Model

Kripke frames are relations in the form of (W, R) where W is a set of possible worlds and R is a set of relations on those worlds.

A Kripke model is in the form (W, R, h) where h is the assignments to the worlds of the model. The function h will be of the form $L \to \wp(W)$ where the language L is the set of all potential assignments. An example Kripke model is shown in Figure 5.1.

Kripke models have the ability to have cyclic relations, where it is possible to reach the same node by a number of steps through other worlds, and also have worlds which relate to themselves. We do not want either of these things in our tree representations.

5.2.2 Monotonicity

We say that a function $f : \wp(S) \to \wp(S)$ is monotonic if for all sets $U, V \subseteq S$, if $U \subseteq V$ then $f(U) \subseteq f(V)$. This means that a monotonic function is entirely non-increasing or non-decreasing. Monotonic functions are required when defining fixed points.

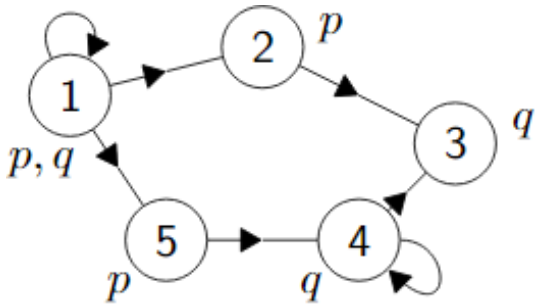


Figure 5.1: Graphical representation of a basic Kripke model

5.2.3 Fixed Points

We have two fixed points of a finite set S and a function f. The greatest fixed point or $GFP(f)$ is the fixed point which has the largest cardinality, the least fixed point or $LFP(f)$ is the one with the lowest cardinality.

We say a point is fixed with regards to a monotonic function f when:

$$f(f(S)) = f(S)$$

From this we can compute the GFP and LFP of a set S as follows:

$$LFP(f) = \bigcup_{n \in \mathbb{N}} f^n(\emptyset) \qquad GFP(f) = \bigcap_{n \in \mathbb{N}} f^n(S)$$

5.2.4 Bisimulations

If we take two Kripke models (W, R, h) and (W', R', h') and let $t \in W$ and $t' \in W'$. A bisimulation between them is a relation $B \subseteq W \times W'$ computed by the following algorithm:

1. $B(t, t')$

For every $u \in W$ and $u' \in W'$ such that $B(u, u')$

2. All atoms that hold at u hold at u'

3. If $v \in W$ and $R(u, v)$ then there is $v' \in W'$ with $R'(u', v')$ and $B(v, v')$

4. If $v' \in W'$ and $R(u', v')$ then there is $v \in W$ with $R(u, v)$ and $B(v, v')$

5.3 Page Comparison Techniques

5.3.1 Hamming distance

The hamming distance is a simple measure of how different two strings of length n are. It is defined as:

$$d(x, y) = \sum_{i}^{n} f(x_i, y_i)$$

where:

$$f(m, n) = \left\{ \begin{array}{ll} 0 & \text{for } m = n \\ 1 & \text{for } m \neq n \end{array} \right\}$$

5.3.2 TF-IDF

TF-IDF is one of the industry standard open source methods of evaluating how important a word is to a document [45]. It is built upon 2 measures:

TF: Term frequency is computed using the ratio of the number of times the term appears with relation to the length of the overall document.

IDF: Inverse document frequency measures how important a term is within a corpus of documents. It first needs to ignore words that appear commonly in written text such as 'is', 'of' and 'that' and then compute the ratio of the total number of documents by the number of documents which contain the term.

Cantina used this method by calculating the most important terms in the page and then assessing whether it appears in the top of Google's search results [53]. If it did not, then it was likely to be a copy of a legitimate page, and thus classified as potentially phishing.

5.3.3 DOM Trees

Often when comparing the similarity of two web pages, it is useful to look at DOM trees. Browsers compute DOM trees as an internal representation of the HTML structure of the page. There is an assumption that if two pages construct the same DOM tree then they will have identical layouts.

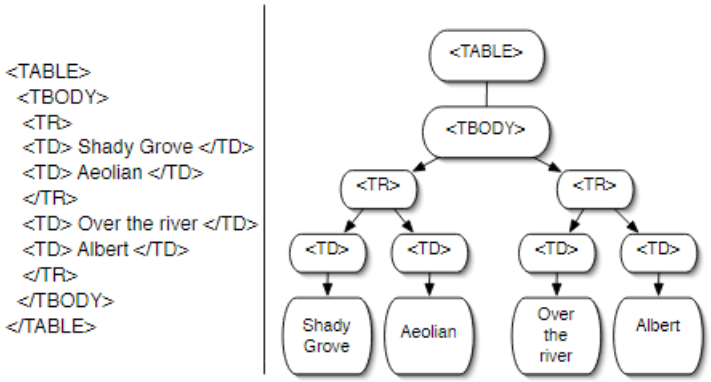


Figure 5.2: Example of a DOM tree representation [40]

Le Dang Nguyen et al. developed an algorithm to compute the similarity of two websites using the extracted DOM trees [29]. They proposed the usage of a genetic algorithm which uses a fitness function when comparing the vertices of two graphs and checking if there is a corresponding vertex in the graph that we are comparing to.

A contrasting method proposed by Qian Cui et al. was to take the DOM trees and compute the proportional distance using tag vectors [12]. The tag vectors count how many of each tag appear in each DOM tree, and then compute the Hamming distance between the counts on each tree. This allows us to compare two similar pages that have been re-arranged by the attacker in order to make the pages seem different.

Although comparing DOM trees is generally a fairly accurate way of computing the similarity between two pages, attackers can make a concious effort to avoid these methods and compute very different DOM trees with similar visual output.

5.4 Image Comparison Techniques

5.4.1 Template Matching

When a template image is to be detected within a larger source image we can use template matching to compare them. The template image is compared to the source image by sliding it. At each point we compare how many pixels in the template image match with the target image and in each iteration we slide one pixel to the right [44].

Template matching is a naive approach which only accepts images that are exactly taken from the target image and that have not been scaled or changed. It is however a very fast approach and is useful in some scenarios. A simple similarity measure for template matching is cross-correlation which computes the sum of pairwise pixels in the image. This is a good metric for the similarity however it can be skewed by bright images that do not correspond with the target.

5.4.2 Affine Transformations

Affine transformations are linear translations that occur with 2D images [1]. Often when comparing images, affine transformations occur intentionally or unintentionally. There are 4 common affine transformations as can be seen in table 5.1.

5.4.3 FasT Matching

FasT matching is a technique developed by Simon Korman et al. for approximate template matching under 2D affine transformations [27]. When using a standard architecture, this technique requires processing time of a few seconds. An example is shown in figure 5.3.

5.4.4 Cascade Classification

Cascade classification is a techinique used primarily in computer vision in order to detect objects or faces within video streams. These classifiers typically use a model

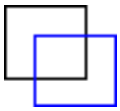
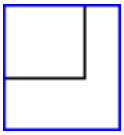
Translation	
Scale	
Shear	
Rotation	

Table 5.1: Table of all possible affine transformations and transformation matrices [1].

which is trained with a series of negative images which do not contain the object, as well as a series of positive images of the object in various positions within different frames.

The term 'cascade' is used since the resultant classifier is composed of several simpler stages which are applied to a region of interest, and accepted if a certain threshold of these stages succeed [7].

5.4.5 Haar Classifier

The Haar feature-based cascade classifier developed by Viola and Jones in 2001 is a rapid detection algorithm with a high degree of accuracy; however the training of the model is a slow process [47]. Motivated by the problem of face detection within live video, the algorithm can process roughly 15 frames per second.

The classifier works by detecting features within a very large feature space of the image, the feature space is much larger than the number of pixels in the image itself.

To reduce the feature space, each weak classifier in the cascade of classifiers is based on a single feature, and on each iteration a set of classifiers is chosen, therefore reducing the feature set. During detection, the small set of features from the training process is loaded and attempted to be detected within the image.

5.4.6 Local Binary Patterns Classifier

LBP classifiers are the more recent approach for object detection within images, it works by thresholding the 3×3 neighbourhood of each pixel and considering the result as a binary number for the pixel [8]. Inside the image, each detected feature can be considered as a cascade of patterns within the pixel radius, which is then added to the trained model.

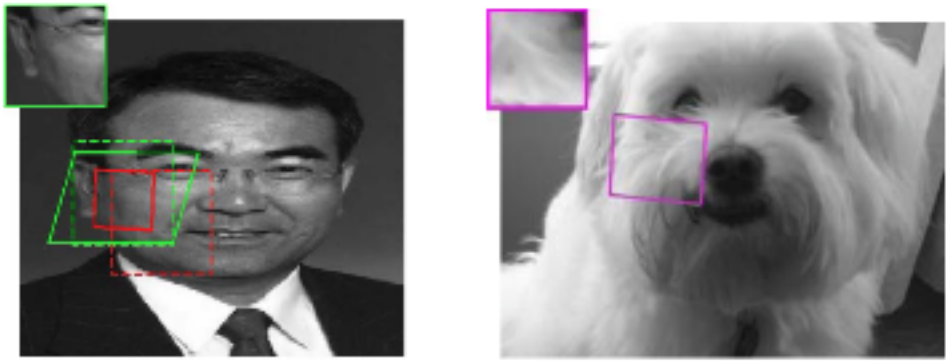


Figure 5.3: Example of FasT matching algorithm results [27]

5.4.7 OpenCV

OpenCV is an open source computer vision library in Python and C++. It was designed for computational efficiency with all of the computationally expensive code written in C++ and interfaces in both Python and C++ through the use of Cython [13][30].

5.5 Web Capture

In order to process pages and output a classification, we need a method of collecting the pages' information for analysis.

5.5.1 WARC

The web archive file format (WARC) is a complete packet capture of a page which aggregates multiple digital resources into a single archive [49]. It is a complete capture of the information a browser retrieves when it visits a specific page. This includes the HTML, status codes and pages which the browser has been redirected to.

5.5.2 Common Crawl

WARC files are the preferred form of storage for Common Crawl. Common Crawl is an open source web crawl with large amounts of crawl data available open source for anyone to analyse. The aim of the project is to make the web open to all rather than large organisations like Google with their resources. There are petabytes of web crawl data spanning over four years in which they have been operating.

5.6 Client Server Architecture

In Python there are two main ways in which we are able to create a client server architecture for communication with the program.

5.6.1 TCP

TCP communication is built into Python using the `socket` library. It is very simple to simply open a port for communication on the local networking protocol. Data can be sent byte encoded over the channel in a very fast manner. However there are issues with ease of use and coupling since a client needs to be built with its own interface and then needs to connect to the server in order to transmit data back and forth.

5.6.2 Flask

Flask is a module within Python which allows us to quickly create a server on `localhost` where we can implement a Rest API for communication. With Flask we do not need to create a dedicated client, we can simply query the browser and read the results with JSON.

6 Implementation

6.1 Candidate URL Collection

As with most machine learning problems, the first issue we encountered was how to collect large amounts of data to train and test the classifier. Lastline as a company deal with WARC files and PNG screenshots, so the classifier was required to be able to process these files.

For the malicious data collection we fetched URLs hourly over the course of 2 weeks from the OpenPhish feed and the IllegalFawn feed [31][25]. These two feeds are provided by independent security researchers who do not publicly disclose their methodologies for obtaining these live phishing URLs, however make the lists public in order to update browser blacklists.

Once these URLs are collected hourly, some pre-processing is done to avoid overloading the analysis infrastructure:

- URLs which have already been analysed are immediately discarded
- Up to two URLs for each public suffix are retained, these public suffixes are obtained from the Python `publicsuffix` library [36]
- A maximum limit of 1024 URLs are retained per hour

For collection of benign data we needed a method of filtering out all malicious data from a large crawl of data. We used Common Crawl for the bulk of the data, however to actually filter out all malicious data from an arbitrary crawl data set would require the classifier which we are trying to create. We assumed that generally popular websites which were in the Alexa list would be very unlikely to be benign so we obtained the Alexa top 1 million list, randomly selected URLs from this list and then we filtered

the Common Crawl data with URLs where the top level domain name was in our set of selected URLs [9][26].

We collected the WARC files for both the benign and malicious URLs in the same way in order to have consistency in the input data for our classifier. To do this, we submitted the URLs to the Lastline sandbox where the URL is visited using Internet Explorer 8 on a Windows 7 machine. The sandbox then monitors the activity performed by the browser and saves the capture of the network traffic generated in a PCAP file which is then converted into a WARC file.

6.2 WARC Parsing

In order to process the data we needed to extract the components from the WARC files into a Numpy array so that they could be fed into the classifier. In order to process each WARC file, we used the Warcio library which provided an iterable array of each record in the file [50].

URL

During the WARC collection process, Lastline's sandbox kept a track of the target URL for each WARC capture and saved it into a CSV file. When processing each file, we obtained the corresponding URL from the CSV file generated by the sandbox.

HTML

Due to the random ordering of the WARC file, it was not possible to simply obtain the HTML content of the first record in the file. We had to find the first HTTP response where the `WARC-Target-URI` corresponds to the URL found in the CSV file. If however, this record happened to be a 301 or 302 redirect, we would then find the corresponding record which it redirected to and extract the HTML from there.

Text

In order to extract the text, we needed to use BeautifulSoup to parse the HTML [2].

We obtained all content which was not contained in `style`, `script`, `head` or `title` tags.

Screenshot

The screenshot was obtained in the candidate URL collection by using Firefox's screenshot terminal interface with the `-screenshot` option. The path to the corresponding screenshot was stored in the array to avoid over usage of memory so that it could be read when needed.

Status

In order to process the status, we obtained the HTTP status from the first record that the URL pointed to. Although we were not necessarily processing the HTML and resources from the initial record relating to the URL, we believed that the status of the first record was the most informative. With a 301 redirect for example, the final record would have a status code of 200 so we would lose information about the initial status if we only considered the status of the final record visited.

Resources

For the resources, we decided to extract the images which were contained within the WARC files as images which the pages would fetch when they were rendering the page. For this we needed to find all records in the archive and extract the bytes of pages which had the HTML content type as `image/png`.

6.3 Pipeline using Cantina and DOM Comparison

We planned to implement a hybrid classifier which combined a variety of different approaches as features in a machine learning algorithm. We decided for an initial implementation to use elements of the Cantina classifier in addition to computing bi-simulations between the DOM trees of new pages to classify and existing known phishing pages [53].

As explained in section 1.1, Cantina's implementation of phishing detection involved

obtaining the textual content of the page, performing the TF-IDF algorithm to find the words with the highest importance, and then assessing if the page appeared in the top n results of a Google search for these keywords.

6.3.1 Pipeline

We needed to initially create the pipeline to process the data and help us to train our classifier using the feature extractors that we were going to create. We created a basic initial pipeline which used a Logistic Regression module to serve as our classifier.

6.3.2 TF-IDF

In order to compute the inverse document frequency of the words in the page we needed a word corpus to compare against. We used the British National Corpus provided by Oxford university which is a 100 million word collection [5]. The words contained in the corpus contain other languages but mainly consist of British English words.

The usage of the British National Corpus brought in biases to the classifier. Pages which did not originate in Britain would use terminology that may not appear as frequently in the corpus and would therefore cause words in certain web pages which are common in other languages to have a high TF-IDF score. Considering also that the word corpus is static, and has been collected over a certain period of time, meant that the corpus would not classify words which may become more commonly used or be created in the future as well.

6.3.3 Google

In order to search Google programmatically, there were three options:

API Usage

Custom Search is a REST API provided by Google to allow website owners to implement search functionality within their own websites. The API would crawl the website which the owner used and then provide a search functionality within that website. Inside the API configuration, there is also an option to include results from the web. We used this to generate an API key which searched the entire web and disabled the site search functionality within the API.

This method was quite fast and reliable due to the Python's interface which Google provided, however there are limitations:

- Google has deprecated the API in the past and it is not currently being maintained with a risk of being deprecated or deleted at any point since Google do not guarantee its uptime.
- Usage limits mean that there are only 100 free queries to the API per day up to a limit of 10,000 queries per day at a cost of $50 per day.
- Google do not provide the same results ordering on their website as they do with the API queries.

Manual Web Scraping

We used the `urllib` package within the Python core modules to query Google and return the HTML result and then used BeautifulSoup to parse the output [2]. This also provided other limitations:

- Google implement abuse and automation blocking techniques such as CAPTCHA and IP address barring [39].

- The method is much slower than the API calls due to the HTML parsing overhead.
- The method would have to be updated if Google were to modify their HTML page structure.

Querying Other Search Engines

There were two other main search engines which we attempted to use instead of Google. Bing was the next option which had an API priced at $7 per 1000 queries with no limit; however this was still far too expensive [3]. DuckDuckGo was also another option who were willing to provide an API for free, however they are an aggregator service for different search engines and as such are unable to provide this service publicly without violating the terms and conditions of the services which they obtain their data from [14].

Faroo is a service who attempted to provide searching as a free service, however their crawling resources are too small and most of their search results are extremely old and therefore would not be useful for our classifier [43].

6.3.4 DOM Comparison

For the DOM comparison, we obtained three examples of malicious and three examples of benign pages and used PyPi's `html-similarity` module to compute the `structural_similarity` and `style_similarity` of the pages [38].

Since we made the assumption about the pages using derivatives of different phishing kits, the `structural_similarity` of the HTML we assumed to be quite similar with phishing pages but not necessarily the `style_similarity`.

Although this seemed to be a good metric some of the time, there tended to be a lot of noise in the results with pages which are stylistically very different but still obtained a high `style_similarity` score. In addition, we were unable to compare against a large number of pages due to the added time complexity of the module.

6.4 Current Pipeline

There were many issues with the previous pipeline which caused issues with scalability. Not being able to query Google as frequently as we wanted was a problem since we needed to train and test our model on around 40,000 examples and we wanted to run long evaluation scripts to test different parameters within our classifier. In addition, the classifier is intended to be used as part of a live classification system which would constantly be run on new data, in this scenario, the API limits provided would be insignificant.

DOM comparison and TF-IDF algorithms caused temporal issues. Our classifier would not be able to classify new examples in the future when the language used had evolved, as well as when the evolution of phishing kits had significantly changed the HTML structure of new attacks. We tried to approach the new classifier such that, as much of the temporal information as possible was able to be obtained from the training data when updating the model, so that the model was able to classify new examples if it was trained regularly.

For each element in our pipeline we have given a robustness score. We have attributed this score to how hard we think it is for attackers to change their pages in order to not be detected.

6.4.1 Item Selectors

We structured the pipeline as a set of feature extractors and a feature union of each of the modules that we wanted to implement. This can be seen in figure 6.1.

With this we implemented a set of item selectors which returned the components which the module needs to run. This is from the set of components extracted during the WARC parsing process.

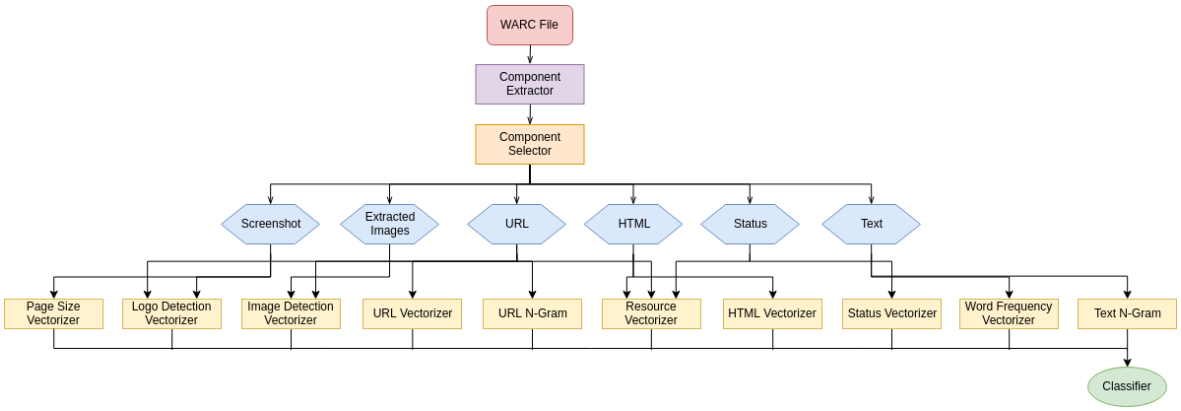


Figure 6.1: Data flow diagram of current pipeline

6.4.2 N-Grams

We implemented n-grams within our classifier in order to detect common patterns on phishing pages. There are common words which occur within the text of a phishing page such as 'password' and 'alert'. N-grams allow the classifier to automatically gain insights of common string combinations.

We have also applied n-grams to the URL in order to detect common words such as 'bank' or 'login'. We varied the value of n in our n-grams in order to find a balance between number of features and information gain.

We have given the n-gram modules a high robustness score since attackers need to use terminology in both the URL and the page textual content in order to lure the user into entering details.

6.4.3 URL Vectorizer

Name of feature	Robustness[1]	Feature Description	Robustness Discussion
URL contains '@' symbol	Low	'@' symbols are common in phishing URLs, they usually occur when the URLs include an email address as one of the GET variables.	To avoid this, the attacker can submit variables by POST methods instead.
Hostname contains '-' symbol	Low	Many malicious URLs use wordpress websites without modification, leaving elements in the URL such as 'wp-admin'. They may also use '-' in their URL if they wish to join to a legitimate hostname, for example 'www.security-hsbc.co.uk'.	The main usage of hyphens is wordpress usage without modification of the path names, to change this the attacker needs to simply modify the path name in the wordpress settings.
URL contains brackets	Low	Brackets appear in URLs where malicious pages upload files which they have previously multiple downloaded, for example doc(1).pdf.	This happens very rarely but it is often a sign of an attacker who has invested a small amount of effort into the page. If the attacker wished to circumvent this, they need to simply rename the files.
Number of Subdomains	Medium	Used as a common method to put a legitimate URL as a subdomain of a problem URL such as `http://bank.barclays.com.bit.ly`.	This can be avoided but is a common technique so it is unlikely for the attacker to do so.

[1] We define robustness as how easy it would be for an attacker to circumvent the particular feature

Name of feature	**Robustness**[1]	**Feature Description**	**Robustness Discussion**
Subdomain contains TLD	Medium	Comparing the subdomain of the URL against the TLD list obtained from the Mozilla public suffix list which is a well maintained up to date list of TLD suffixes [37].	As we can see in section 3.2.4, this is a common technique used by attackers and as such is a robust feature to detect.
Path contains TLD	Medium	Comparing the path of the URL against the TLD list obtained from Mozilla public suffix list [37].	If the attacker was unable to create a subdomain on the host, this would be the next most favourable option to try and trick the victim.
Path contains '=' symbol	Medium/Low	GET requests are common within phishing attacks because they are easier to process than POST or PUT requests.	There is no way to avoid using '=' symbols if GET requests are being used and since they are common, it is a robust feature to detect.
Hostname contains number	Medium/Low	Commonly occurring when hashed information is stored in the URL.	Often used as a technique for attackers to hide sensitive information in a GET request.
URL contains cyrillic characters	Medium/Low	Cyrillic characters can look very similar to English characters, there have been phishing attacks in the past where attackers used these characters to register domains where the URL looks exactly the same as the official page [34].	Although many major browsers have taken preventative measures to tackle this issue, this feature is often a strong signal that the attacker may be using such methods.

Name of feature	**Robustness**[1]	**Feature Description**	**Robustness Discussion**
Shortened URL	Medium/Low	Although users have legitimate reasons to use URL shortening services, it is also a common technique to mask the malicious URL from the victim. Using a frozen list of URL shortening services is the only way to detect the usage since there is no way our classifier can learn which new services are to be created in the future.	Since we are forced to use a frozen list, the attacker is able to find a URL shortening service that we have not already detected if they wish to circumvent this feature.
Target in subdomain	Medium/Low	Check if a frozen set of targets exists in the subdomain of the URL. Targets are in a frozen list but get updated using the PhishTank API [35].	We use n-grams for learning new words that are being used as a more robust feature. However checking for a target in a list of words is a useful feature with less robustness but also has a very low performance cost.
Trigger in path	Medium/Low	Check if a frozen set of trigger words exist in the path of the URL.	If the list is obtained, these triggers can be easily avoided however the trigger words are often words that attackers need to use in order to fool their victims.

Name of feature	**Robustness**[1]	**Feature Description**	**Robustness Discussion**
Target in path	Low	Check if a frozen set of targets is in the path.	It is not common for the targets to be in the path and it is easily avoided by the attacker, however the feature is very strong when detected.
Changed port	Medium	Check if the port has changed, it is common for attackers to compromise the DNS records of legitimate websites and redirect traffic on an unused port to the malicious page.	If the attacker is able to modify the DNS, they will often use a different port on the website they gained access to so this is a robust feature.
Misspelled target in URL	Medium	This can often be useful as a separate feature than simply detecting the target in the URL.	Misspelled targets is one of the most common ways in which attackers attempt to fool users that they are on the legitimate page. Therefore, in addition to detecting the correctly spelled URL, the misspelling is also important to detect.
Length	Medium	Number of characters in the URL.	In addition to the other features, malicious URLs are generally longer than benign URLs, since they attempt to hide extra information using methods described above.

Name of feature	Robustness[1]	Feature Description	Robustness Discussion
Uses HTTPS	Medium	Although it is now more common for phishing pages to employ HTTPS according to the latest report by APWG, in order to seem legitimate, for most websites it is not worth the effort for the attacker to install an SSL certificate [32].	To circumvent this feature, attackers can use SSL more in the future, however with the short time to live of most phishing pages and that SSL certificates are already freely available today with letsencrypt, attackers are unlikely to start using HTTPS more in the future [28].
Size of query	Low	This is the size of the query in the page, for example in `https://www.google.com/q=test` the query length would be the length of the word 'test' which is four.	Queries can easily be avoided by attackers however it still seems to be a strong feature for phishing URLs.

6.4.4 Word Frequency Vectorizer

Using the TF-IDF algorithm on the textual content of the page was a useful heuristic in the previous version of the pipeline, however we were reliant on the usage of the static word corpus. To solve this we decided to use the combination of the text from all of the pages within the training set as our word corpus. Therefore, when we trained our model on new examples, the word corpus would be updated. In addition the corpus would be relevant to the pages which we are classifying upon, meaning that the language bias introduced by using the British National Corpus would be removed.

We believe the robustness of this vectorizer to be much higher than the previous implementation since the words that are important to a page are not from a single source. For this reason, if the attacker wanted to circumvent this feature, they would have to find words that are uncommon in web pages today rather than querying the British National Corpus and using words which have a low ranking.

6.4.5 Status Vectorizer

We wanted to investigate the HTTP status of the page and use this as a feature for the classifier. We initially considered using an n-gram for the status however this caused a large number of features to be created which was not useful for our classifier. Therefore we decided to extract the base status number from the HTTP status line and create features for common status numbers we deemed to be relevant.

The robustness of the status vectorizer is very high because there is no way for the attacker to manually spoof the HTTP status response code that we obtain. We are able to detect re-directions which are very common in phishing pages and combined with the HTML, we can further detect if a page is attempting to fake an HTTP status and returning a different page. For example we can detect if the page is in fact found with a status code of 200 but the website is attempting to serve a fake 404 page instead.

6.4.6 HTML Vectorizer

Name of feature	Robustness	Feature Description	Robustness Discussion
Page uses a favicon	Very Low	Basic feature which checks if the page has a favicon present or not.	Although attackers typically do not use favicons, it is simple for the attacker to add one and thus circumvent this feature.
Number of forms	Low	We used this feature rather than the boolean value of whether forms are present on the page because we believe that phishing pages generally only use a single form element and other benign pages may use multiple forms.	The attacker can create invisible forms in order to increase the number of forms on the page.
Number of password fields	Low	Malicious pages tend to contain a single form with a single password field.	In the same manner as number of forms, the attacker can add invisible password elements as well.
Number of iframes	Medium/Low	Iframes are HTML components which are generally not used frequently anymore but still remain prevalent in old phishing kits.	This feature is trying to detect pages using older phishing kits directly without modification. However, if the phishing kit is reliant on iframes, for the attacker to change this would involve them having to rewrite that part of the kit to work without iframes which is time consuming.

Name of feature	Robustness	Feature Description	Robustness Discussion
Scripts to HTML ratio	Medium/Low	We assume that phishing pages generally have a low HTML page content, since they require only to convince and capture the user's data. However, many kits also use scripts heavily to capture as much information as possible. These pages will have a high script to HTML ratio.	If the attacker wishes to reduce their ratio, and they are using a lot of scripts, they would need to add copious amounts of invisible HTML code which could then become a feature itself to detect. It is also unlikely that the attacker would try and prevent being detected with such a feature.
Number of tags	Medium/Low	This feature indicates the size of the page ignoring the amount of text inside the tags.	In order to change this the attacker would also have to create a lot of invisible tags, as in for the scripts to HTML ratio feature.
Size of page	Medium/Low	This gives us the number of characters in the HTML page. This feature in conjunction with the number of tags allowed us to detect pages which have a large amount of text inside tags such as paragraph tags.	In order to change this the attacker would have to add text or padding into the page to increase the length which they are unlikely to do if they wish to seem legitimate as a page.

6.4.7 Resource Vectorizer

The resources on the page, such as the form actions, and the links of the page, are good items to investigate. We believe this since phishing pages generally will have pages which are hosted on a different server, with links to the legitimate page, as well as potentially links to their page. They will also need to direct the form POST action, to a page which they are controlling, in order to store the phished credentials. For this analysis we investigated the URL, HTML and the status. The URL was important to know whether the links and form actions pointed to the same host, or alternative hosts, and inspecting the status also allowed us to detect, with the resources, whether the page was performing an HTTPS upgrade, or a redirect.

We examined all the forms and links on the page and returned whether the form actions were empty, relative paths to the current page, pointed to a page on the current domain or pointed to external domains. We also went on to check whether the form actions pointed to a single host, or pointed to multiple different hostnames, since we believe that phishing pages generally want to direct all traffic to websites which they control. Once the user has logged in and they have phished the credentials, they would also usually redirect the user to the target page so as to not arouse suspicion.

6.4.8 Screenshot Vectorizer

Whenever phishing sites wish to imitate a page, they need to try and convince the victim that they are on the page of the target. In order to do this, they would need to display the logo or some distinguishing visual signal of the target on the page. We hypothesised that if we were able to detect the target logo on the page, we would then be able to verify that the hostname of the URL matched with the hostname of the logo.

We initially needed to collect a list of common targets so that we could select the top targets to detect. To do this we took the top 20 targets from PhishTank by querying their data sets and obtaining the targets which appeared with the highest frequency

[35].

Once we had the top targets, we downloaded the first 10 images from Google image search results for these target names. We had to manually edit each image after this, in order to obtain a high-quality image, with the logo on a transparent background, and removing all textual content from the logo images.

We attempted a series of different image detection methodologies to try and detect the logo in the page. The fastest approach we found to be template matching, however, this did not work with affine transformations, and the logos on the page were usually different sizes and resolutions to the reference logos we had obtained.

Our next step was to attempt to use feature matching in the hopes that logos which appeared on the page would detect a larger number of features than others. The results can be seen in figure 6.2. As we can see in this figure, when trying to compare three different logos with an AT&T phishing page, the PayPal logo matched the most key-points within the image, and the Microsoft logo matched very few key-points. We also noticed that the Microsoft logo only had four potential key-point matches, and as such for every page which we attempted to match it against, the number of matches returned was significantly lower than any other logo. There were also many matches with the logo and alternate points on the screenshot, usually the text, which did not correspond to the logo on the page. We could have attempted to correct for this using key-point clustering, however, the volume of direct matches between the target logo and the logo on the page, did not prove consistent enough to warrant this approach as being worthwhile.

The next attempt was to use FasT matching, however there was no readily available implementation on the internet that worked well with Python [27]. We also attempted to use feature matching with FLANN which keeps feature points matched which have a lot of nearest neighbours [18]. We thought that this could filter out our feature matching points so that we could keep only the points which corresponded to closely

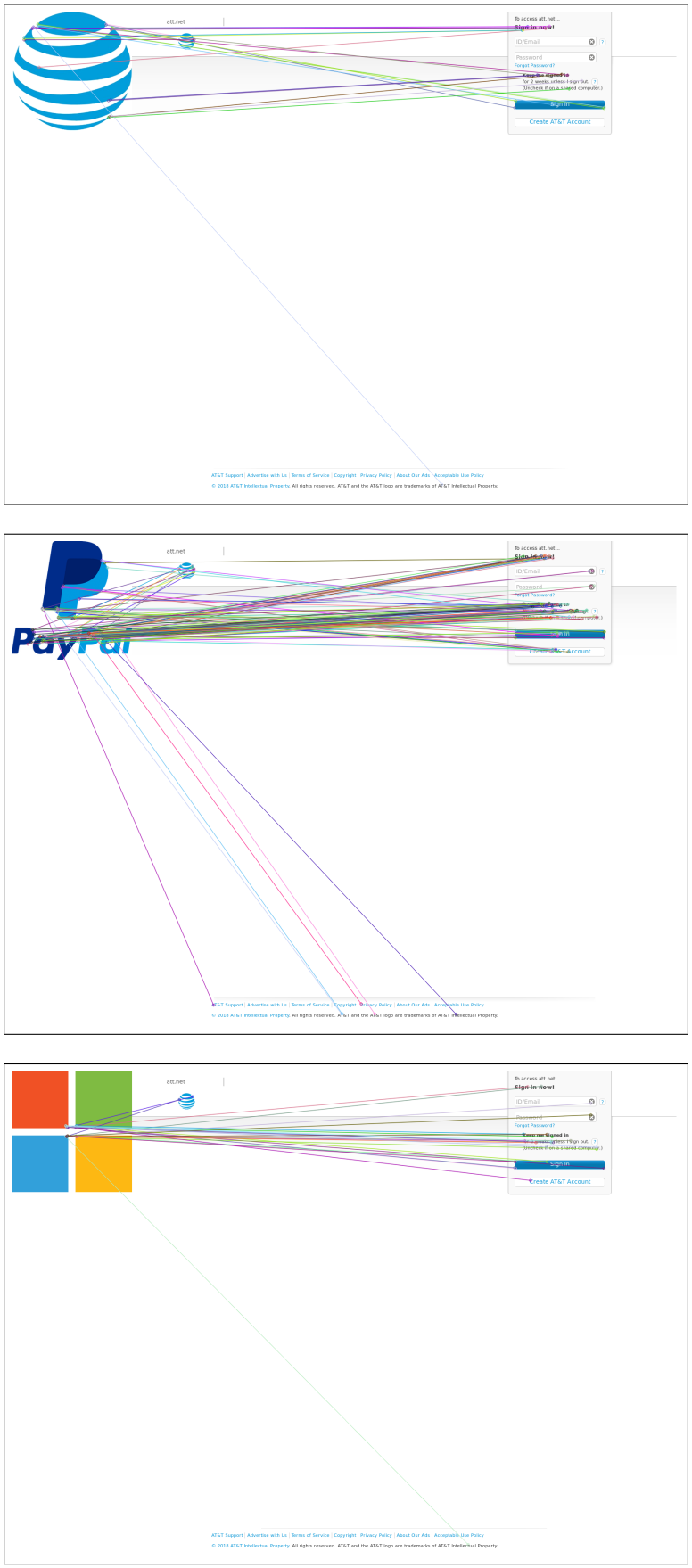


Figure 6.2: Comparison of feature matching with AT&T, PayPal and Microsoft logos on AT&T phishing page

grouped ones, as the logo on the page. This also did not work, because the feature detections within our screenshots were not strong enough that all the matches were filtered out as can be seen in figure 6.3.

We then tried to use a cascade classifier, we trained the cascade classifier with the logos we obtained. To do this the classifier needs to learn with positive and negative images. For the negative images we obtained a set of random images from the internet, these images could contain any other graphics, however, they could not contain any logos from the ones which we wanted to train. We used OpenCV to generate a set of positive images with the logo, applying a set of transformations and then superimposing it onto the negative images using `opencv_createsamples` [30].

The training of each cascade classifier with the set of logos took approximately five hours to train per logo. The cascade classifier however worked relatively well for most logo detection as can be seen in figure 6.4.

We did have issues with a large number of trained logos however. The Netflix logo worked extremely badly as can be seen in figure 6.5. We believe this may be due to the structure of the logo and that it does not have as many defining features. In the future careful logo selection and testing must be done before adding a trained cascade to the pipeline.

We also noticed that many of the screenshots were very long and would incur a performance hit if we applied the cascade classifier to the entire image. For this reason, we cropped the height of the screenshots to be at maximum equal to their width. We did this so that we would reduce the time taken for the cascade classifier to try and find the logo as we made the assumption that the main logo of any page would be situated at the top of the page.

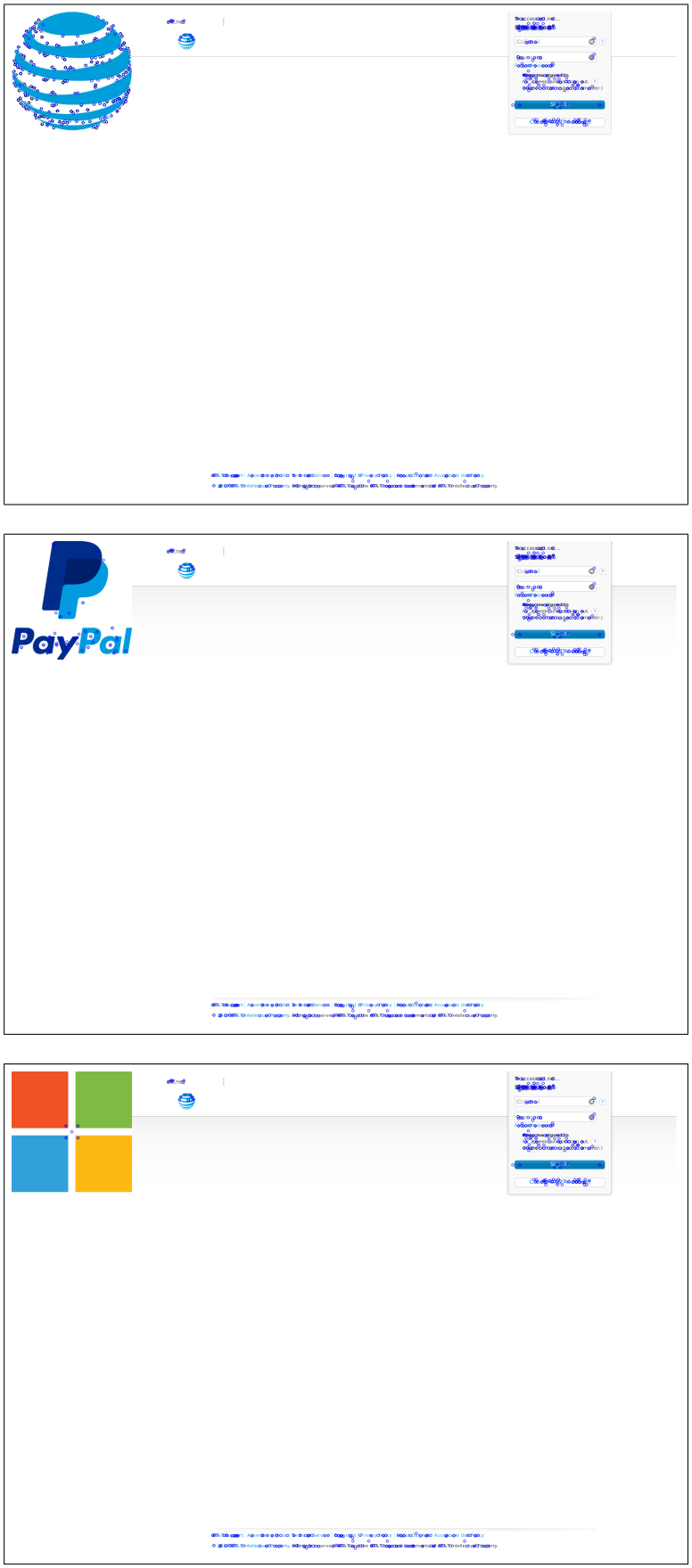


Figure 6.3: Comparison of FLANN matching with AT&T, PayPal and Microsoft logos on AT&T phishing page

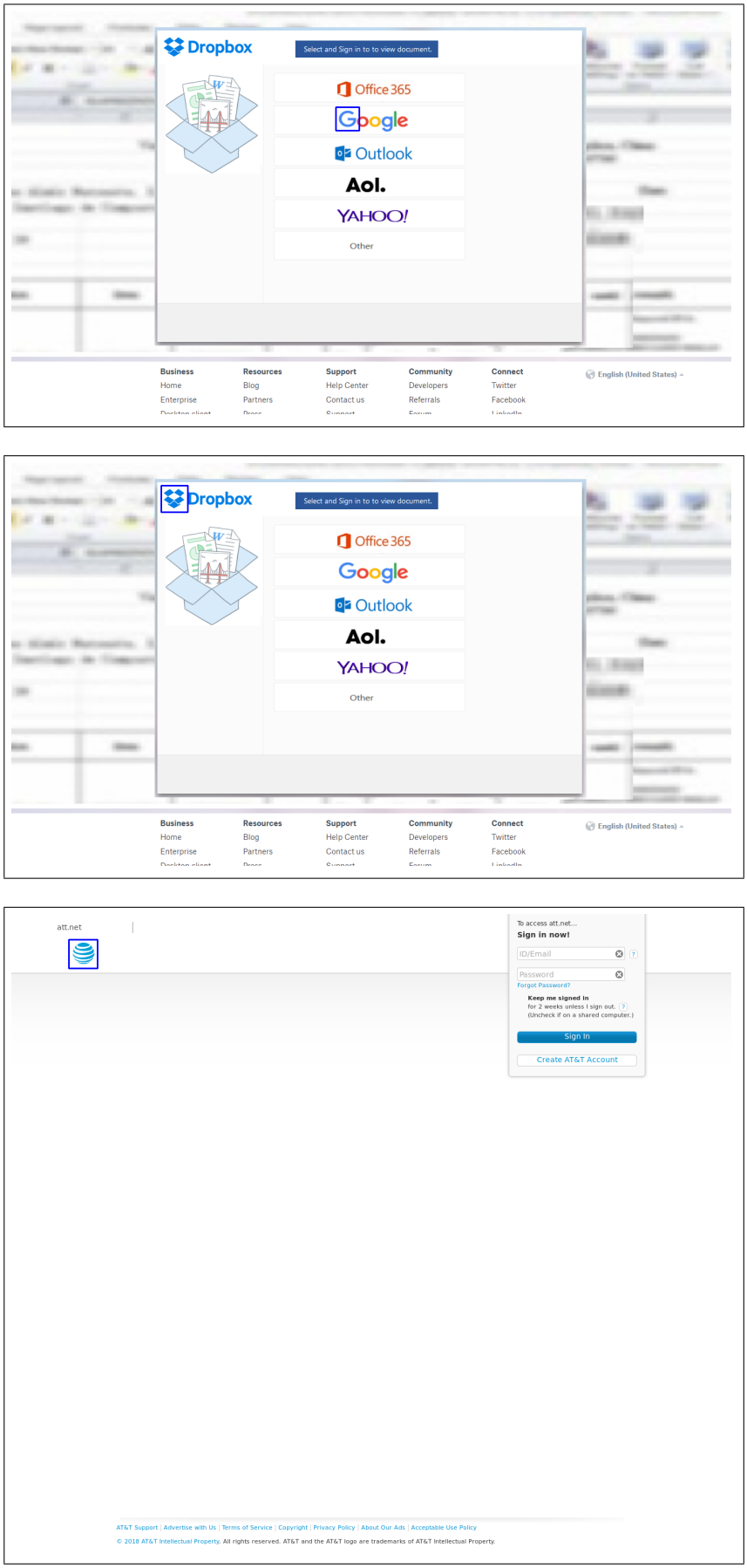


Figure 6.4: Successful cascade matching with Dropbox AT&T and Google logos

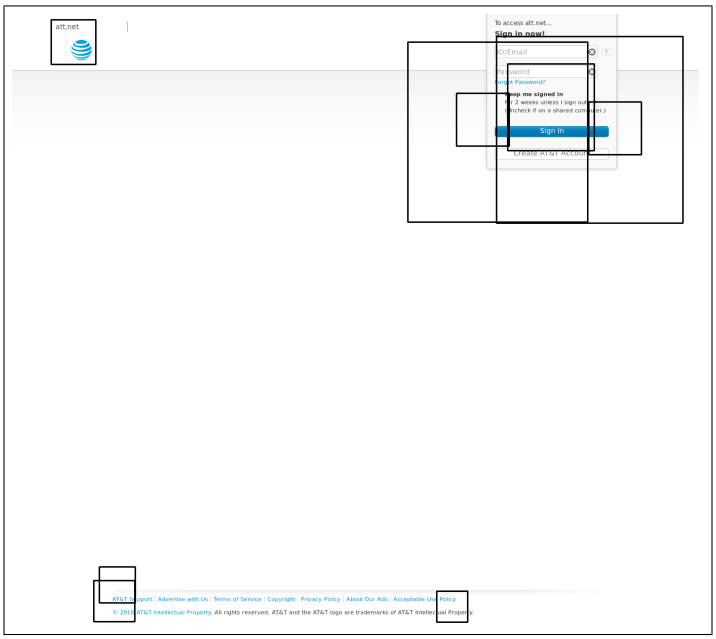


Figure 6.5: Issues with cascade trained on Netflix logo with AT&T phishing page

In addition to the issues with cascades which were not working very well, such as the Netflix logo, we noticed a large number of false positives which were much harder to avoid. The logos were being identified on the page by patterns which appeared to be very similar to stylistic elements of the logo, however they were not the actual logo. We can see some of these issues in figure 6.6.

We realised that, when the logo was correctly identified on the page, the bounding box of the matched image would contain pixels and colours which were similar to the logo we were matching and when the match was fitting on styles, it would sometimes be very different colours. We decided that, to add a further check with the matched area whether it was in-fact the logo or not would be sufficient. For this reason, we extracted the portions of the screenshot which had been detected by the logo and attempted to match them with the logo. We re-sized them in order to make them the same dimensions as the logo, flattened both 2-dimensional images into 1-dimensional and then applied the `compare_ssim` function from scikit-image to compare them. We achieved a high accuracy second layer of match filtering by accepting if the `compare_ssim` function returned a negative value.

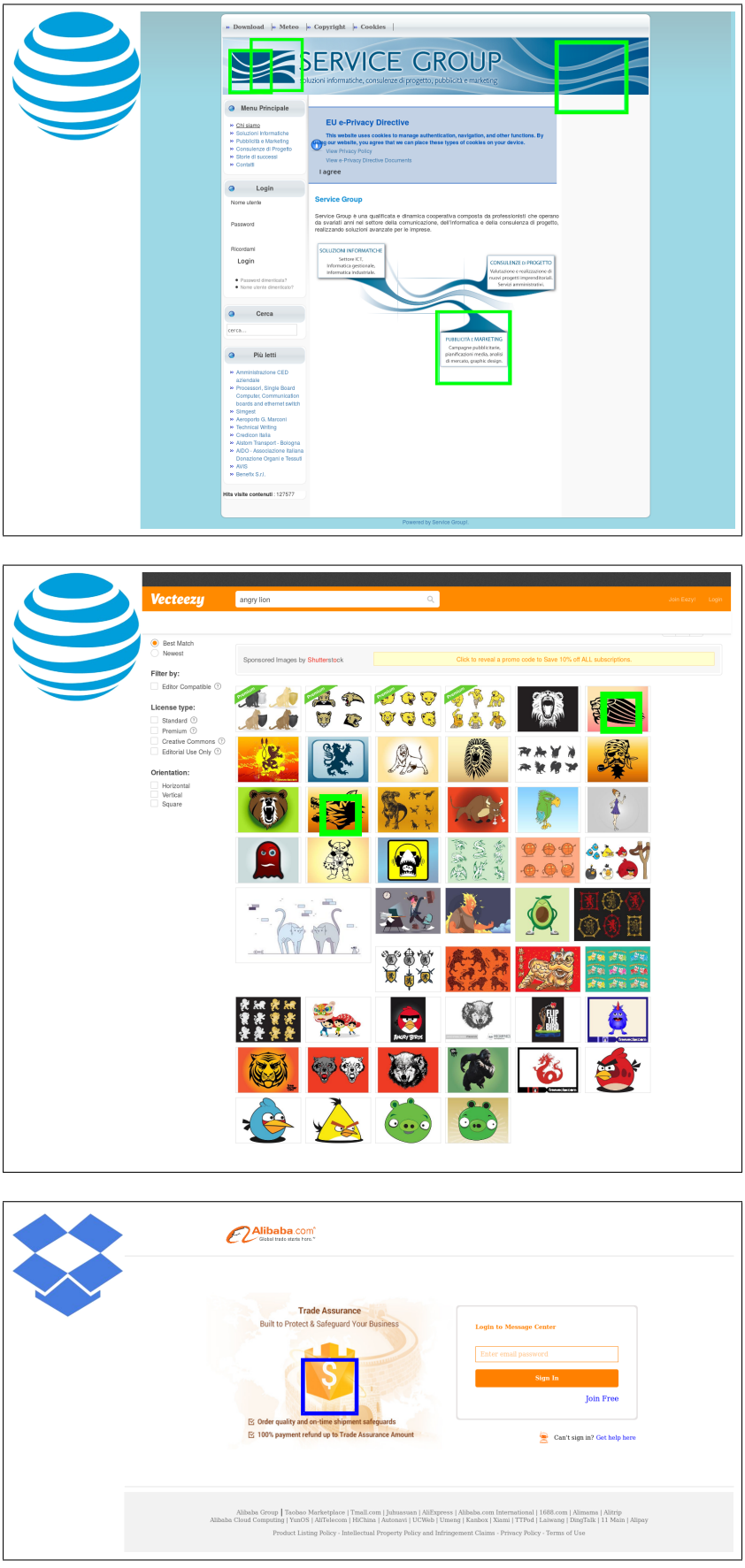


Figure 6.6: Issues with cascade matching of logos matching with patterns on the page but not of the target logo

6.4.9 Image Resources Vectorizer

We realised three main issues when using the screenshot vectorizer:

- Not all pages had screenshots which meant that we were not applying this technique on a large number of pages
- We had to crop these pages for performance so we were losing potential information with the bottom part of the screenshots
- By scanning the entire page we are wasting a lot of time scanning parts of the page which would not contain any logos

To fix these issues we decided to run the cascade classifier on the image resources extracted from the WARC files. We did this because we believed that any logos which are on the page would be loaded into the page and therefore the browser would have to fetch these images manually.

6.5 Client Server Architecture

In order to avoid the overhead of fitting the pipeline to the trained model before returning a classification, we implemented a client server architecture using Flask which loads the trained model when it is started up [51]. Since the time taken to classify a WARC file is significantly less than the overhead of the client-server architecture, we allowed the list of files to be classified to be sent via JSON and a JSON object to be returned, with a mapping between the name of the WARC file and the classification which it has been given, thus allowing us to classify multiple WARC files per request.

6.6 Batch Script Execution

As with many machine learning projects, there were many long running scripts which we had to run in order to train and evaluate our classifier. We had to use many different architectures in order to run our classifier. We used external computational power in

order to handle the large memory requirements of some of the intensive graphical computation and to avoid slowing down our machine during development.

6.6.1 VMware EXSi

We were provided access to the external computation cluster by Lastline. The cluster made use of a VMware EXSi hypervisor which allowed us to create virtual machines remotely and run our scripts on there [48]. Since we were able to create our own environment, we were able to clone the development setup of Linux as we were running on our machine and then we were able to use the Python virtual environment to run our code without compatibility issues.

6.6.2 SSH

We were also able to run our computation on batch servers provided by Imperial College London. The advantage of running these on here was the large disk space allocation so we were able to copy all of our training and test data which allowed us to run batch training algorithms over the course of multiple days. We used tmux sessions in order to keep processes running in the background over SSH [46].

One of the issues we encountered was transferring large files over to the batch computing cluster. We had around 40 gigabytes of training and testing data. We were able to use SSH tunnelling and the ability with RSync to partially copy files over and restart an interrupted process which allowed us to copy the files without being concerned about interruptions in internet connection or power issues [41].

6.6.3 Super Computers

We also employed the use of super computers to speed up some of the processes that we were running. We created a conda environment which we used to deploy our processes out to the super computer cluster [10]. There were various different queues for super computers which had better infrastructure and wall time availability but

required longer wait times for the job to be run.

6.7 Issues

6.7.1 HTTPS Upgrades

During the collection of our WARC files, we submitted the URLs we collected to the LastLine sandbox to output the entire traffic capture for the session. However when website perform the HTTPS upgrade, they signal to the browser to create a new encrypted session for communication.

For this reason, in our WARC files, all of the web traffic after the redirection in an HTTPS upgrade was encrypted and we were unable to analyse it.

6.7.2 OpenCV Memory Leak

We rely on OpenCV quite heavily for the image detection portions of our classifier [30]. The implementation of OpenCV in Python which we use interacts with native C++ code using Cython for the performance critical code. While regular Python is good with garbage collection, there is a leak in the underlying code of OpenCV. We opened a GitHub issue at `https://github.com/opencv/opencv/issues/11559` but this is yet to be resolved.

We temporarily patched this issue by manually deleting variables in Python as well as calling the garbage collector using the function `gc.collect()` to free the memory, however this problem did sometimes still occur on our personal development box. For this reason, we had to also use the external computing cluster sometimes with greater RAM resources to work around the memory leak until it is fixed by OpenCV.

7 Evaluation

In this section we will discuss the performance and accuracy of our classifier and compare it against other related work in the field.

7.1 URL Collection

URL collection is one of the most common areas where bias can be brought into a machine learning model. For the malicious URL collection that we performed, we believe that the main issue was that we were learning over already classified examples. Since we were constantly polling a set of sources from researchers in the field, a lot of the pages we classify have already been marked by other automated classifiers. For this reason, we may be training and testing our model on pages which have stronger features than some of the more difficult pages to classify.

For the benign data collection there are also potential issues with introducing bias in the popularity of the pages. The Alexa list is a descending list of the most popular pages on the internet, when we take the top 1 million list we are considering only pages which are popular on the internet [26]. We also have no guarantees that these pages are truly legitimate, there is a possibility that pages which are malicious may be successful enough to be in the Alexa top 1 million; however we believe this to be unlikely. Also the bias of pages outside of the top 1 million is not a huge concern to us as we are more interested in classifying examples from daily user traffic which will tend to be on popular pages.

The submission of pages to the Lastline sandbox in order to be classified can also bring about some further bias in the data. The browser and operating system will be included in the user agent and may cause the page to act differently with regards

to this. We also introduce a temporal element since we performed the data collection over a period of 6 weeks, interspersed with gaps, where no data was collected. For the benign data we were constantly running freshly detected phishing pages which reduced the risk of them already being taken down. However for the benign pages, they were collected from the latest archive of crawl data in March 2018, and the capture was done during the same 6 week period during May 2018, by extracting the URLs and submitting them to Lastline's sandbox.

R. Gowtham et al, a well cited machine learning approach to phishing classification used similar methods for URL collection, using lists of popular pages from Google, Netcraft and Alexa for their benign page collection and Phishtank, Millersmile and Reasonable-Phishing Webpages List for their malicious collection [19].

In addition, H. Zuhair et al created a respected literature review of machine learning methods to tackle phishing classification in November 2017 [55]. This can be seen in figure 7.1. From this we can see that all of the most popular phishing classifiers use similar methods for URL collection. For this reason, although our methods may introduce biases, we believe they are the best methods available today for URL collection.

Study / Issues	[14]	[15]	[16]	[17-18]	[21]	[22]	[19-20, 23]
Machine Learning	SVM	SVM, LR, BN, DT, Adaboost	SVM/TSVM	SVM	Neural Network	SMO, LR, RF, NB	SVM, C4.5, RF, JRip
FBC design	Single	Ensemble	Single	Single	Ensemble	Single	Single
No. of Features	12	15	6	17	17	15	212
Type of Features	Hybrid features	Hybrid features	Textual/visual features	Hybrid features	URL/Textual features	URL features	URL/Textual features
New Features	Not	8 new features	Not	3 new features	Not	5 new features	Not
Data Set (Phish/Not Phish)	325/200	8118/4883	200/200	1764/700	800/600	1416/1462	48,000/48,000
Data Set Source	PhishTank, CastleCops, Millersmiles	PhishTank, Alexa, 3Sharp	PhishTank, GoogleWhite	NetCraft, MillerSmile, GoogleTop, Alexa	PhishTank, Millersmiles,	Chinese Data Archives	PhishTank, DMOZ
Detected Phish Types	Redirecting, Login Form, Homepage, e-Business, English	Redirecting, Login Form, e-Business, Social Networking, English	e-Commerce Login Forms, English	e-Commerce, Login Forms, Redirecting, English	e-Commerce, Login Forms, Redirecting, English, French	e-Business, Chinese	e-Commerce, English, French, German, Italian, Spanish, Portuguese
Overall Performance*	TP 97.33% FP 1.45%	TP 92%; FP 1.4%	Accuracy 96.4%; Miss Rate 3.5%; Mistake Rate 5.4%	TP 99.6%; FP 0.42%	Accuracy Rates 91.31-94.7%	Accuracy Rate 95.83%	Accuracy Rate 99%, FP 0.004%
Demerits	Typical features; Few feature; Small data set, No feature selection; Inactive learning, No Adaptation, Time consuming	Imbalanced data; Typical features; Few features; No feature selection; No adaptation; Time consuming	Imbalanced data; Typical features; Few features; Small data set, No feature selection No adaptation	Typical features, Few features; Small data set, No feature selection, No adaptation	Typical features, Few features; Small data set, No feature selection, Inactive learning, No adaptation	Typical & few features, One feature category, Imbalanced data; Inactive learning, No adaptation	Typical features, One feature category, No features selection, No adaptation
Notes	FBC=Feature-Base Clasifier, *SVM =Support Vector Machine; LR=Logistic Regression; BN=Bayesian; DT, C4.5, and JRip are types of Decision Tree Classifier; RF=Random Forest; NN=Neural Network; SMO=Sequential Minimal Optimization, NB=Naïve Bayes.* *Overall performance outcomes are adopted as they presented in the related works.						

Figure 7.1: Comparison of various machine learning approaches to phishing in the literature today [55]

7.2 Data Structure

After ingesting the WARC files during our data loading process, we obtained the data splits as seen in 7.2. We split the data set by allocating them based on the first character of the hashed name of the WARC files.

In total our test set is 23.04% of our complete data set. We also had a larger proportion of our data set as malicious than benign. The proportion of our test data set which was malicious files was 71.80% compared to 63.77% for our test set.

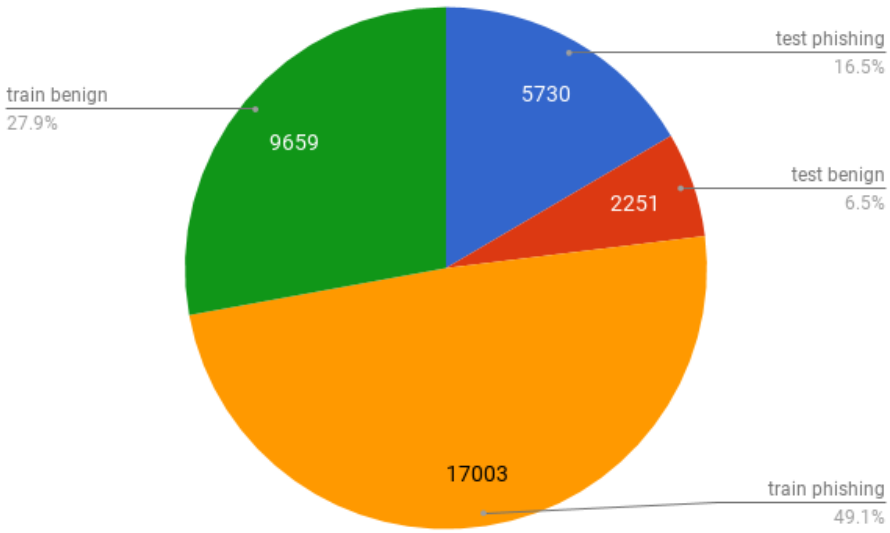


Figure 7.2: Distribution of our ingested benign and phishing WARC data into test and train splits

7.3 Ensemble Selection

In order to select which ensemble we wanted to use, we had to keep the default parameters in our classifier and test our model against the three main types, the results can be seen in figure 7.3.

As we can see, Random Forest is by far the best ensemble for our data. We believe this is because there are various distinct feature sets for which separate decision trees are much better suited. Since we have a large feature set, ensembles like logistic regression would find it difficult find a line of best fit through all of the points.

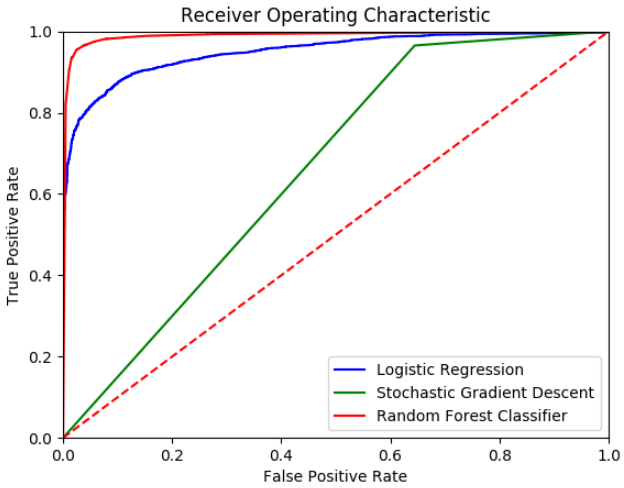


Figure 7.3: ROC curve displaying the relative accuracy of the three main ensembles with our classifier

7.4 N-Grams

We use two N-grams in our project, one for the URL and one for the body of the page. Typically one would select a value of N=2 or N=3. Since there are 256 ASCII characters, in a 2-gram, the maximal number of features would be $256^2 = 65536$ whereas with a 3-gram it is $256^3 = 16777216$.

For this reason it was important to verify which value of N was appropriate. If we chose N=3 we would gain more information however we would have a very large feature set which could lead to over-fitting as well as a longer classification time. As can be seen in figure 7.4, the 3-grams gained a slight information gain on the 2-grams however they took far longer to classify and combined with the other features in our classifier we were afraid of overfitting with too many features.

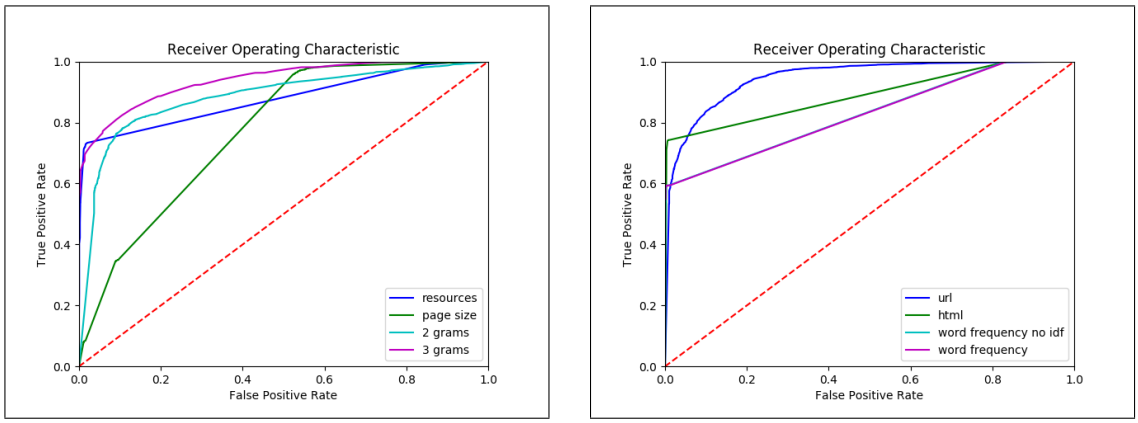


Figure 7.4: ROC curves displaying the information gain from each of the feature extractors in our classifier

7.5 Word Frequency

For the word frequency vectorizer, we evaluated how well it worked in our classifier with and without using the IDF part of the TF-IDF algorithm as can be seen in figure 7.4. We did this by enabling the `no_idf` option within the base `TfIdfVectorizer`. As we saw here, the line on the ROC curve when applying IDF and when applying TF only was exactly the same. We can ascertain from this that the computation of the IDF has no effect on the overall classifier. We can attribute this to two main potential reasons:

- Usually malicious pages are small and therefore will have a small amount of textual content on the page, for this reason, the high TF words on the page will probably be the malicious words and the application of the IDF algorithm would not change the selected words.

- Many times, phishing pages have such a small textual content on the page if they simply have a login form, that all of the words on the page will be submitted to the classifier for analysis and none of the words will be ignored due to the low IDF score.

7.6 Logo Detection

We applied logo detection in two manners, attempting to detect the logo within the screenshot of the page and attempting to detect the logo within the image resources on the page. We trained and ran our model using a Random Forest classifier with the detection in the screenshot only, and in the extracted images only, our results can be seen in figure 7.5.

As we can see here, we obtain a small information gain over our test set with detecting logos within screenshots but no information gain with detection of logos in extracted images. The issue we discovered during our implementation was that the logo detection system was extremely sensitive to false positives in the cascade classifier. This is because, if there is a single model within our set of logos which performs in the same way as the Netflix model shown in figure 6.5, then every page would have the Netflix logo detected on it and none of them would actually derive from the Netflix hostname. This would cause the feature values for the screenshot detection to be the same for each example in our data set and the classifier would be unable to learn any information. We attempted to rectify for this by only including a few, well verified logos within our classifier during the evaluation of these components.

This leads us onto the second potential issue with this method. Having very few trained models of our logos within our classifier caused issues. To work the best, the classifier needs an exhaustive list of trained cascades for all logos that may appear in certain pages. If any logo is detected, the classifier then checks if the hostname corresponds to the expected hostname of the logo in question. If however, we manage to detect logos in the page without actually detecting the logo of the main page then we would provide a classification leaning toward a malicious page when the page is in fact benign. In figure 7.7 our classifier detected logos situated within the social media buttons on the page but did not detect the main BBC logo and the page was therefore misclassified as malicious.

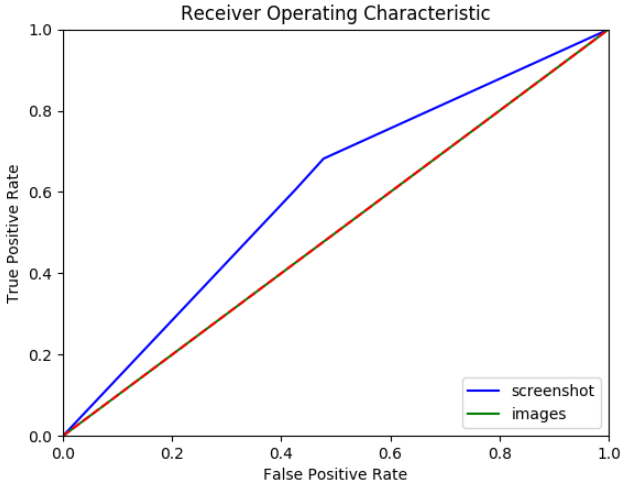


Figure 7.5: Comparison of the performance of detection of logos within the screenshot and the extracted images in our classifier

The other main issue which we tried to correct for during the implementation of this classifier in section 6.4.8 was that the cascades would match on similar patterns which occurred within the screenshots of the page which did not correlate to the logo in question. We tried to correct this by doing a further comparison with the colours of the logo and the colours in the matched portion of the page which worked well however we still had false positives when the colours also happened to coincide with the logo.

Overall, we believe that the module has a lot of potential to be an extremely strong metric as we have proven in the various positive examples shown in the implementation. Unfortunately, due to time constraints of the project, we were unable to try and solve the issues that still remain with the module, and its high sensitivity to false positives means that it did not produce desirable results.

7.7 Feature Importance

In order to evaluate which features in our classifier are the most important, we are able to obtain the information gain that each feature has in the classifier. The results are

shown in appendix A.2. It is important to note that, due to the large feature set that the n-grams produce and the fact that these features would not be easy to analyse, we have not included the n-grams in our feature importance table.

As we can see, most of the information gain that we obtain is with static analysis of the URL, status and the HTML. We saw some information gain from the logo detection methodologies in the screenshot but no information gain over the training set for the logo detection within the extracted images. It is evident that, if, even with the small number of logos and the issues we faced with the component as described in section 7.6, we were able to gain a feature which performs better than some of the other features we have, then when the issues are solved, it could be a very strong feature.

Surprisingly, one of the highest variance features in our classifier is the page height. This was a very simple metric but it proves our theory that phishing pages are generally short as compared to non-phishing pages which may be longer. We also have many of the other well-known features from the literature which attain high importance. In addition to this we gain high importance with features from the resource vectorizer we proposed ourselves.

We also saw zero variance for many of the status numbers, this is to be expected, because even some of the common status numbers may have no bearing on whether the page is malicious or not. This may also be down to statistical bias which we obtain from our URL collection, since the crawl data and phishing sources would not submit results with the other status numbers such as a 500 server error. These codes are also less common with popular domains which have development teams to ensure their pages are always active.

7.8 Classification Rate

Our classification results for our classifier over the test set are displayed in table 7.1. As we can see here, we are achieving an accuracy of 96.50%. We can also see that the false

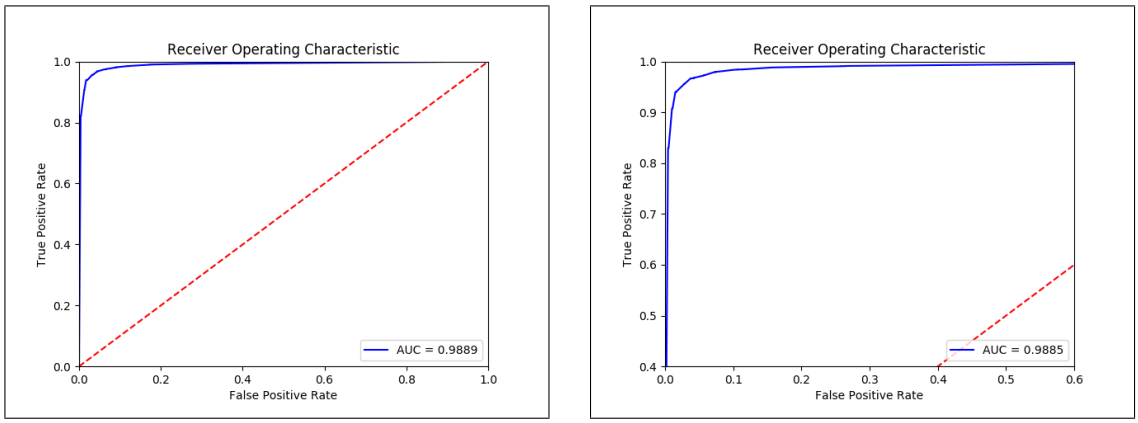


Figure 7.6: Full and zoomed in images of the ROC curve of our classifier.

positive rate with respect to the number of pages classified as phishing and the false negative rate with respect to the number of pages classified as non-phishing are similar.

Also, from the ROC curve for our classifier in figure 7.6, we can see that we obtain an extremely high true positive rate for a false positive rate of 0.05 and for a false positive rate of 0.20 we have obtained a true positive rate of almost 1.0. However, we only reach a true positive rate of 1.0 with a false positive rate of around 0.6.

		Test result	
		Non-phishing	Phishing
True classification	Non-phishing	2159	92
	Phishing	187	5543

Table 7.1: Performance of our classifier over our test set

7.9 Performance Evaluation

Performance was a big consideration in the project. As discussed in section 7.6, analysing the logo in the image resources rather than the screenshot saw a marked improvement in the performance of the overall classifier.

Model training is something that will be performed weekly, the time to train will be similar to the time to classify for each training example. We also have an initial overhead when classifying for communicating with the server, for this reason we allowed an array of WARC files to be submitted to the server, returning an array of classifications for each file.

Our current classifier takes on average 0.35 seconds per example to train and 2.5 seconds for each query to the server. We also have an overhead of initialising the OpenCV module which takes approximately 30 seconds to initialise however this overhead does not exist for subsequent queries in the same session [30].

7.10 Comparison with Current Methods

Proofpoint Emerging Threats is a leader in open source and commercial malware detection [17]. They have created a set of rules which allow users to identify a basic level of phishing detection very quickly over a wide range of pages [15]. We ran their rules over the WARC files in our test set and obtained the confusion matrix in table 7.2.

Safe Browsing is Google's ever growing blacklist of pages which are deemed to be malicious [42]. They provide an API for checking if a series of URLs are classified as malicious or not. The API provides 10,000 requests in a 24 hour period however they allow up to 500 URLs to be classified in a single call. For this reason they can not be used for live querying however at the end of each day we can query up to 5 million URLs. We have included the results of safe browsing on our test set in table 7.3.

		Test result	
		Non-phishing	Phishing
True classification	Non-phishing	2251	0
	Phishing	5595	135

Table 7.2: Performance of applying PhishPunch rules over our test set [15]

		Test result	
		Non-phishing	Phishing
True classification	Non-phishing	2251	0
	Phishing	3918	1812

Table 7.3: Performance of querying Google Safe Browsing with our test set [42]

As we can see with both Emerging Threats and Safe Browsing, they have zero false positives but high rates of false negative. This is because Emerging Threats find indisputable signals of phishing and Safe Browsing needs to consistently ensure that pages submitted to their blacklists are all malicious since their lists are being used as in-browser blocking for Google Chrome.

Our classifier works better than many of the solutions that are current state of the art as seen in figure 7.1 and far better than Emerging Threats and Safe Browsing, however it does not perform the best. We do believe that due to the constraints that we had to adhere to with our classifier and by achieving better results than most of the commonly used methods in literature today, our methodology does improve the state of the art. These constraints include performance constraints as well as not having access to the DNS records of the page and inability to query a search engine such as Google.

7.11 Overfitting

One of the most common concerns in machine learning problems is overfitting. When a model is overfit, it performs well on the test set however it does not perform well when deployed live. This is mainly due to the fact that the model has become accustomed to bias when collecting the training and testing data. We used two techniques to test for overfitting in our classifier.

7.11.1 K-Fold Cross Validation

Cross validation is useful to evaluate whether the model is simply good at classifying on our test set specifically or if it is able to classify the entire data set at the same rate. This is generally good at checking whether we have spent too long trying to extract features from our misclassifications and therefore become overfit on our test set. We used k-fold cross validation with k=10. The results are shown in table 7.5.

As we can see here, we are achieving fairly similar accuracy for each fold of the cross validation as we are for the results on the test set within ±1.5%. We can attribute the slight loss in accuracy for some of the rounds to the fact that each iteration in the cross validation is training on a smaller data set. Also some of the splits may contain less useful training examples, hence it is able to learn less information. Regardless, we are clearly still able to classify the rest of the data set almost as well as our test set.

Round	TN	FN	TP	FP	Accuracy
1	1100	86	2232	47	96.16%
2	1117	49	2244	55	97.00%
3	1140	71	2185	69	95.96%
4	1106	73	2218	67	95.93%
5	1123	75	2215	51	96.33%
6	1145	68	2201	50	96.57%
7	1169	61	2187	47	94.54%
8	1100	74	2215	75	95.67%
9	1141	96	2155	72	95.12%
10	1174	72	2156	62	96.08%

Table 7.5: Result of 10-fold cross validation of our classifier over the entire data set

7.11.2 Balance

We also have an imbalance in our data set, our benign data set is 38.38% of the size of our malicious set. Imbalances can cause bias within the classifier because the classifier can become too accustomed to seeing a certain proportion of the data to be a certain

classification. In order to remove bias, we attempted to reduce the malicious testing and training data set to the size of the benign data set and then evaluate our model on that. We obtained the results as in table 7.6.

With the balanced data set, we achieved an accuracy of 96.00%. This value falls right in line with the accuracy of the classifier with the cross validation results in table 7.5. For this reason we believe that we are not overfitting on imbalanced data sets.

		True classification	
		Phishing	Non-phishing
Test result	Phishing	2187	116
	Non-phishing	64	2135

Table 7.6: Result of our classifier with a balanced data set

7.12 Misclassifications

We believe it was important to investigate which misclassifications we were getting in each iteration of the classifier. For this reason, during each run of our evaluation we stored the classification, URL, HTML and screenshots of the pages which we had misclassified. These are investigated below.

7.12.1 URL

When inspecting the URLs which had been incorrectly classified as benign, the first thing we noticed was the distinct lack of features which most malicious URLs would have. We are satisfied in this case that our classifier is correctly classifying the malicious pages which exhibit clear signs that the URL is malicious. Some examples of malicious URLs which have been misclassified are below:

- `https://forums.neswangy.net/forumdisplay.php?f=5`
- `http://www.locarmix.com.br/plugins/content/souchkldekl`

- https://rciinvoice.com/office/12/12/error.php

From initial inspection of these URLs, it is very difficult to ascertain any features that make it different from a benign URL from a human perspective. We did, however find a small set of URLs in our larger misclassified set which contained features we could pick up on as below:

- https://eastnootropics.com/update-P.ayPal.com/
- https://steamcommunituy.com/1523540873090/mn4zqaprqmnyej6gvoy/ku...
- https://theatretech.co.za/ws/node/v3/WELLSFARGO-ACCOUNT-UPDATE.html

These URLs had features which we had missed which we could have very easily extracted also such as 'WELLSFARGO' in the path, long URLs with 'steamcommunity' misspelt and 'paypal.com' in the path which has dots added. These features would easily be picked up if we had increased our target set to include these targets, however we did not want to do this since we would be overfitting on our test set.

For the URLs which we had misclassified as malicious, most of these URLs did not exhibit malicious characteristics, this would suggest to us that the majority of these classifications had occurred due to issues with the other parts of the page rather than the URL analysis specifically. We did encounter a few examples which were benign URLs but had characteristics of malicious URLs, these can be seen below:

- https://www.usgs.gov/staff-profiles/craig-d-allen?logstash-usgs-pw%3Apalladium_root_topics=&logstash-usgs-pw%3Apalladium_root_publication_year_date=&sort=&page=2
- http://www.bbc.com/culture/story/20180312-the-racist-message-hidden-in-a-masterpiece?ocid=ww.social.link.twitter
- https://www.heavy-r.com/out.php?member=homemade-voyeur.com&url=http%3A%2F%2Fwww.homemade-voyeur.com%2Ftube%2Fvideo%2Fwaterpark-exposure-xwmYyVF9fNN.html

All of the URLs above are very long. The first and the third URLs also heavily use percentage encoding which is very common in malicious URLs. In addition the second and third URLs contain what looks like other URLs inside the path, this is a very common tactic in malicious URLs and is probably why these have been flagged in our classifier. We also have found that a lot of the benign URLs which have been misclassified are from the adult industry such as in the third URL. The industry tend to use language which is more akin to malicious URLs in order to entice their users to click on the URL. We have included a full list of misclassified URLs in appendix A.1.

7.12.2 HTML

The vast majority of misclassified pages had no HTML code within the body of the WARC files. This is something that could be attributed to the WARC collection process. For the pages which have been incorrectly classified as benign and have HTML, we have found that the structure is usually very similar to legitimate pages.

The HTML page for `http://khatsaz.com/bankofamerica.com.login.account.update/1ab5b207453ce26b2c1474c0714db8da/signInScreen.go.html` is structured very similar to a benign page. Although the page is malicious it does not have a form on the page and is not trying to obtain information from the victim. The page also has most resources retrieved from the same domain as the page is hosted on and has a small script to body ratio.

A page which had been misclassified as malicious is `http://tnaflix.com/robots.txt`. This page is simply a 301 redirect, however it is a permanent redirect so we did not immediately detect the redirected page in our WARC file. The only HTML we were able to analyse was from the redirection page. For this page we were unable to ascertain much information from the URL either so it was very difficult to classify correctly.

7.12.3 Screenshot

With screenshot detection, as discussed in the implementation section 6.4.8, we had issues with many of our logos not creating cascade models which worked well. The manual work required to obtain the target logos and editing them in addition to the time taken to train the models was also a lengthy process. For these reasons, we only created models for very few logos in our training process. This gave us low detection rates with our classifier since we only had models of a few logos which matched a small proportion of the pages.

We also had issues with benign page misclassifications due to the screenshots. Since we do not have a complete list of logos, we had issues with the page as in figure 7.7. In this screenshot, we have the login buttons for the social media accounts but we did not have the trained model for the BBC logo in the top corner. We detected the logo, however the hostname of the target URL did not match the hostnames collected for any of the images detected within the screenshot, and the page was incorrectly classified.

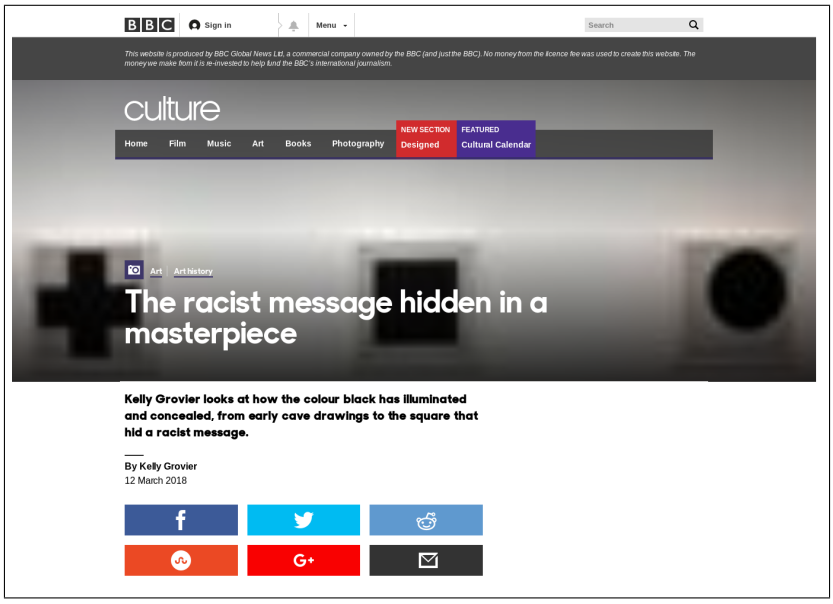


Figure 7.7: Example of a misclassified page where we detected the logos within the social media buttons however we did not detect the main BBC logo of the page.

8 Conclusion

8.1 Summary

We have developed a classifier which is able to classify a page in 2.3 seconds with 96% accuracy which falls in line with the current state of the art. We have been able to future-proof our classifier by implementing various methods such as learning the word corpus for TF-IDF directly from the training data as well as using n-grams so that the classifier can detect common letter sequences that pages use in their text as well as their URLs.

Further, we used computer vision methodologies in order to detect logos within web pages quickly and accurately, using methods which have not been applied before to this problem. Although we had some issues towards the end of the project with successfully integrating this functionality, we believe that given more time, we could have resolved the issues that we were facing. We have also gained confidence from the results of the classifier, that once these issues are resolved, they could accelerate the accuracy of our overall classifier to rates better than the state of the art today.

By creating an easily extendable architecture, new methodologies can be added in order to improve the classification in the future when they are discovered. We have extracted elements from the entire page such that new modules can analyse the HTTP status, externally loaded resources, the HTML, the screenshot and the URL.

We thoroughly evaluated our classifier and all potential biases to the point where we are confident that we are not overfitting on our data and it will continue to work well in the future to classify new pages.

8.2 Future Work

Through this research, we have been able to identify opportunities to improve the accuracy of the classification.

8.2.1 Further Resource Analysis

When we were investigating externally loaded resources from the WARC files, we analysed only PNG images which had been loaded onto the page. There are many other resources that are loaded as well such as scripts and stylesheets. In addition, these resources have information such as status codes, and which websites they are loaded, from which would be interesting to analyse.

Through our analysis of the external resources, we found that a lot of pages which loaded images and stylesheets from external places, may often load the images directly from the target domain. It would have been interesting to be able to check as a feature, that if the logo was detected inside the image which was downloaded, whether the source of the image had matched any of the URLs which we had obtained as part of logo capture.

8.2.2 Searching

During the implementation of the first pipeline, we were achieving a lot of success when we were finding the top TF-IDF words on the page and subsequently querying Google for them. Due to the usage limitations of Google's search API and the lack of a suitable replacement where the usage allowances matched with our requirements, we believe that this could be added as a separate feature into our current pipeline again and it would help us to classify many more pages. All of the implementation for this module has already been completed, we would simply need a suitable source to query.

8.2.3 Safe Browsing

Google Safe Browsing has a requirement such that all URLs in their databases must be malicious, they are unable to tolerate any false positives [42]. They also have an additional advantage from ourselves that, in addition to using automated classifiers, they also use human judgement where users flag pages which they have visited and are malicious.

For this reason it could be useful to take examples from our classifier which are marked as benign but are flagged by Safe Browsing, and then training our models with these examples. This would be useful because we would then have more training data constantly being collected at the end of each day and our classifier would be able to strengthen on the features which it is currently not identifying correctly.

8.2.4 DNS Checking

One of the things that we were unable to do in our current classifier is to check the DNS records of the pages which we were analysing. As we discussed in section 3.3.6, the DNS contains a lot of valuable information with regards to phishing classification, since attackers do not generally register malicious pages in the same way that legitimate pages are registered. Unfortunately, there was not a suitable API which allowed us to query this information, as well as the desire to keep all of the analysis offline, however this can easily be added as a further module in the future.

8.2.5 Live Model Training

The current structure of our classifier is that we train the model on a large data set and then we use the trained model to classify new examples. However, when the classifier is deployed, we will have various sources of confirmed pages which are legitimate or benign, either from the various phishing classification sources, from Google Safe Browsing or from the other classifiers which are deployed at the company [42]. We could develop a method where we can keep feeding our model newly classified examples and update

the model on the fly rather than having to re-train over the entire data set again.

8.2.6 DOM Comparison

We did DOM comparison with our classifier in the first pipeline, however it did not work out too well due to the method of training the machine learning model. We were not able to perform direct comparisons with other DOM trees during the training, and we were forced to extract features instead. There was some initial progress, however, with offline DOM comparison between different phishing pages, and we believe that our hypothesis about DOM trees of phishing pages being closely related is true. It would be interesting if there was a way to extract the features of the DOM tree into binary features which we could input into our classifier and allow this to try and detect derivatives of other phishing kits being used.

8.2.7 Automatic Logo Training

We have already developed a method to periodically query PhishTank for the latest most popular targets of phishing attacks, automated scripts which take a target, download the top n images from Google images, and training scripts which generate the model from the logo [35]. The only issue which is stopping us from creating a fully automated solution for fetching logos, training their cascades, and adding their cascades to our pipeline, is the manual selection and editing of the logo to fit standards. We believe that either finding a source of plain logos which we could use, or creating a script which can automatically select and remove extra data from images and giving the images transparent backgrounds, would fully automate this portion of the pipeline as well.

8.2.8 Script Analysis

Scripts on the page, and from externally loaded files, are commonly used to track and fingerprint the way victims may use the page, as well as performing actions in the browser to convince the victim to enter their information. However, it remains a part

of the page which we have not investigated yet. The only heuristics we took from the scripts currently were their size in relation to the rest of the page and HTML code. Many more features can be extracted from these and added to the classifier.

Bibliography

[1] *Affine Transformation - MATLAB & Simulink*. URL: https://uk.mathworks.com/discovery/affine-transformation.html (visited on 01/20/2018).

[2] *Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (visited on 06/16/2018).

[3] *Bing*. URL: https://www.bing.com/ (visited on 06/16/2018).

[4] *Bitcoin.com | Bitcoin News and Technology Source*. URL: https://www.bitcoin.com/ (visited on 06/16/2018).

[5] *British National Corpus*. URL: http://www.natcorp.ox.ac.uk/ (visited on 06/02/2018).

[6] *Building a Spam Filter Using Machine Learning*. URL: https://www.booleanworld.com/building-spam-filter-using-machine-learning/ (visited on 05/30/2018).

[7] *Cascade Classification — OpenCV 2.4.13.6 documentation*. URL: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade%7B%5C_%7Dclassification.html (visited on 05/18/2018).

[8] Jo Chang-yeon. *Face Detection using LBP features*. 2008. URL: http://cs229.stanford.edu/proj2008/Jo-FaceDetectionUsingLBPfeatures.pdf.

[9] *Common Crawl*. URL: http://commoncrawl.org/ (visited on 06/16/2018).

[10] *Conda - Conda Documentation*. URL: https://conda.io/docs/ (visited on 06/14/2018).

[11] Marco Cova, Christopher Krügel, and Giovanni Vigna. "There Is No Free Phish: An Analysis of "Free" and Live Phishing Kits". In: *WOOT*. 2008.

[12] Qian Cui et al. "Tracking Phishing Attacks Over Time". In: *Proceedings of the 26th International Conference on World Wide Web - WWW '17*. New York, New York, USA: ACM Press, 2017, pp. 667–676. ISBN: 9781450349130. DOI: 10.1145/3038912.3052654. URL: http://dl.acm.org/citation.cfm?doid=3038912.3052654.

[13] *Cython*. URL: http://cython.org/ (visited on 05/30/2018).

[14] *DuckDuckGo — Privacy, simplified.* URL: https://duckduckgo.com/ (visited on 06/16/2018).

[15] *EmergingThreats/phishpunch*. URL: https://github.com/EmergingThreats/phishpunch (visited on 06/14/2018).

[16] *Essentials of Machine Learning Algorithms (with Python and R Codes)*. URL: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ (visited on 05/18/2018).

[17] *ET Intelligence*. URL: https://www.proofpoint.com/uk/products/et-intelligence (visited on 06/14/2018).

[18] *Feature Matching with FLANN — OpenCV 2.4.13.6 documentation*. URL: https://docs.opencv.org/2.4/doc/tutorials/features2d/feature%7B%5C_%7Dflann%7B%5C_%7Dmatcher/feature%7B%5C_%7Dflann%7B%5C_%7Dmatcher.html (visited on 06/16/2018).

[19] R. Gowtham and Ilango Krishnamurthi. *A comprehensive and efficacious architecture for detecting phishing webpages*. Feb. 2014. DOI: 10.1016/J.COSE.2013.10.004. URL: https://www.sciencedirect.com/science/article/pii/S0167404813001442.

[20] Xiao Han, Nizar Kheir, and Davide Balzarotti. "PhishEye". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. New York, New York, USA: ACM Press, 2016, pp. 1402–1413. ISBN: 9781450341394. DOI: 10.1145/2976749.2978330. URL: http://dl.acm.org/citation.cfm?doid=2976749.2978330.

[21] Shuichiro Haruta, Hiromu Asahina, and Iwao Sasase. "Visual Similarity-Based Phishing Detection Scheme Using Image and CSS with Target Website Finder". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. IEEE, Dec. 2017, pp. 1–6. ISBN: 978-1-5090-5019-2. DOI: 10.1109/GLOCOM.2017.8254506. URL: http://ieeexplore.ieee.org/document/8254506/.

[22] Ian Hodkinson. *499H | Faculty of Engineering | Imperial College London*. URL: http://www.imperial.ac.uk/computing/current-students/courses/499H/ (visited on 01/20/2018).

[23] *Hook, line, and sinker - A closer look at a sophisticated phishing kit | Proofpoint United Kingdom (GB)*. URL: https://www.proofpoint.com/uk/threat-insight/post/hook-line-sinker-sophisticated-phishing-kit (visited on 12/31/2017).

[24] *How author Michael Wolff got his 'fly-on-the-wall' access to the Trump White House*. URL: https://eu.usatoday.com/story/news/politics/2018/01/04/how-author-michael-wolff-got-his-fly-wall-access-trump-white-house/1003673001 (visited on 06/01/2018).

[25] *IllegalFawn*. URL: https://twitter.com/illegalfawn (visited on 06/13/2018).

[26] *Keyword Research, Competitive Analysis, & Website Ranking | Alexa*. URL: https://www.alexa.com/ (visited on 06/16/2018).

[27] Simon Korman et al. "FasT-Match: Fast Affine Template Matching". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2013, pp. 2331–2338. ISBN: 978-0-7695-4989-7. DOI: 10.1109/CVPR.2013.302. URL: http://ieeexplore.ieee.org/document/6619146/.

[28] *Lets Encrypt*. URL: https://letsencrypt.org/ (visited on 06/03/2018).

[29] Le Dang Nguyen, Dac-Nhuong Le, and Le Trong Vinh. "Detecting phishing web pages based on DOM-tree structure and graph matching algorithm". In: *Proceedings of the Fifth Symposium on Information and Communication Technology - SoICT '14*. New York, New York, USA: ACM Press, 2014, pp. 280–285. ISBN: 9781450329309. DOI: 10.1145/2676585.2676596. URL: http://dl.acm.org/citation.cfm?doid=2676585.2676596.

[30] *OpenCV library*. URL: https://opencv.org/ (visited on 05/03/2018).

[31] *OpenPhish*. URL: https://openphish.com/ (visited on 06/13/2018).

[32] *Phishing Activity Trends Report 4Q 2017*. URL: http://docs.apwg.org/reports/apwg_trends_report_q4_2017.pdf (visited on 06/16/2018).

[33] *Phishing Schemes Are Using HTTPS Encrypted Sites to Seem Legit | WIRED*. URL: https://www.wired.com/story/phishing-schemes-use-encrypted-sites-to-seem-legit/ (visited on 12/31/2017).

[34] *Phishing with Unicode Domains*. URL: https://www.xudongz.com/blog/2017/idn-phishing/ (visited on 06/13/2018).

[35] *PhishTank | Join the fight against phishing*. URL: https://www.phishtank.com/ (visited on 12/31/2017).

[36] *PublicSuffix*. URL: https://pypi.org/project/publicsuffix/ (visited on 06/13/2018).

[37] *PublicSuffix*. URL: https://publicsuffix.org/ (visited on 06/13/2018).

[38] *PyPI - the Python Package Index*. URL: https://pypi.org/ (visited on 06/16/2018).

[39] *reCAPTCHA: Easy on Humans, Hard on Bots*. URL: https://www.google.com/recaptcha/intro/v3beta.html (visited on 06/16/2018).

[40] Angelo P. E. Rosiello et al. "A layout-similarity-based approach for detecting phishing pages". In: *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*. IEEE, 2007, pp. 454–463. ISBN: 978-1-4244-0974-7. DOI: 10.1109/SECCOM.2007.4550367. URL: http://ieeexplore.ieee.org/document/4550367/.

[41] *rsync*. URL: https://rsync.samba.org/ (visited on 06/14/2018).

[42] *Safe Browsing – Google Safe Browsing*. URL: https://safebrowsing.google.com/ (visited on 01/18/2018).

[43] *SeekStorm - Search as a Service*. URL: https://seekstorm.com/ (visited on 06/16/2018).

[44] *Template Matching*. URL: https://docs.adaptive-vision.com/current/studio/machine%7B%5C_%7Dvision%7B%5C_%7Dguide/TemplateMatching.html%20http://docs.adaptive-vision.com/4.7/studio/machine%7B%5C_%7Dvision%7B%5C_%7Dguide/TemplateMatching.html (visited on 01/20/2018).

[45] *Tf-idf __ A Single-Page Tutorial - Information Retrieval and Text Mining*. 2010. URL: http://www.tfidf.com/ (visited on 01/18/2018).

[46] *tmux/tmux: tmux source code*. URL: https://github.com/tmux/tmux (visited on 06/14/2018).

[47] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. IEEE Comput. Soc, pp. I–511–I–518. ISBN: 0-7695-1272-0. DOI: 10.1109/CVPR.2001.990517. URL: http://ieeexplore.ieee.org/document/990517/.

[48] *VMware ESXi: The Purpose-Built Bare Metal Hypervisor*. URL: https://www.vmware.com/uk/products/esxi-and-esx.html (visited on 06/14/2018).

[49] *WARC, Web ARChive file format*. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml (visited on 05/29/2018).

[50] *warcio*. URL: https://pydigger.com/pypi/warcio (visited on 06/16/2018).

[51] *Welcome | Flask (A Python Microframework)*. URL: http://flask.pocoo.org/ (visited on 06/16/2018).

[52] Colin Whittaker, Brian Ryner, and Marria Nazif. "Large-Scale Automatic Classification of Phishing Pages." In: *NDSS*. The Internet Society, 2010. URL: http://dblp.uni-trier.de/db/conf/ndss/ndss2010.html#WhittakerRN10.

[53] Yue Zhang, Jason Hong, and Lorrie Cranor. "CANTINA: A Content-Based Approach to Detecting Phishing Web Sites". In: *WWW07*. 2007.

[54] Yue Zhang et al. "Phinding phish: Evaluating anti-phishing tools". In: *Human-Computer Interaction Institute* Paper 76 (Jan. 2006). URL: http://repository.cmu.edu/hcii/76/.

[55] Hiba Zuhair and Ali Selamat. "Phishing classification models: Issues and perspectives". In: *2017 IEEE Conference on Open Systems (ICOS)*. IEEE, Nov. 2017, pp. 26–31. ISBN: 978-1-5386-0731-2. DOI: 10.1109/ICOS.2017.8280269. URL: http://ieeexplore.ieee.org/document/8280269/.

A Appendices

A.1 Misclassified URLs

URL	Classification
http://homesbylistorti.com/websites/2/images/Boa/19f1abc4c465912113ddocs/	benign
http://www.alguaciles.cl/assets/tinymce4/js/themes/modern/DHL2016/DHL/tracking.php?	benign
https://www.aryahotel.com/core/.index.html?/fpi2vw09hjogd7jhmv/	benign
http://activeliving.com.au/Dropbox/	benign
https://mikyc.com/wp-includes/random_compat/byte_home/Tempcontrol/	benign
https://whatsmyfoodintolerance.com/blog/wp-content/plugins/pb-mium/inc/flap/ca7a4ee6eab83e5d0128ef3320996f8b/	benign
https://www.kienfat.com.au/wp-content/plugins/DropNewVasion/Fresh/index.html	benign
http://thegadgetisland.com/upd/Counter/index.php?email=	benign
https://web.kompyte.cloud/by_timestamp/1505291139084/config/default/intermediate/https://www.nab.com.au/	benign
https://valleynco.com/cgi/mail/login/	benign
http://palominoroad.com/wp-content/themes/redline/lunch/cameo.php	benign
http://rosemakeupartist.com/jur/image.htm	benign
https://theatretech.co.za/ws/node/v3/WELLSFARGO-ACCOUNT-UPDATE.html	benign
https://www.qinetwork.com.br/como-melhorar-os-processos-da-sua-empresa-com-google-docs/	benign
http://jomjb.com/adobeaa/JSK/Adobe/adobeaa/9d47d25f5fa9a0d9c0b4a67949e6dd6c/index2.htm	benign
https://eastnootropics.com/update-P.ayPal.com/	benign
http://popodoosmartenglish.com.vn/images/class/index.html	benign
https://www.yastk.com/savefiles/kutuphane/wait.html	benign
https://rciinvoice.com/office/12/12/error.php	benign

URL	Classification
https://steamcommunituy.com/1523540873090/mn4zqaprqmnyej6gvoy/kujnfkdmm4j1yziya3r/lymtbeo6xartw6w1plzc/5smypbuvl87w9ufg5kd?openid.mode=checkid_setup&openid.ns=http://specs.openid.net/auth/2.0&openid.identity=http://specs.openid.net/auth/2.0/identifier_select&openid.claimed_id=http://specs.openid.net/auth/2.0/identifier_select&openid.return_to=https://skull-bets.io&openid.realm=https://skull-bets.io	benign
https://forums.neswangy.net/forumdisplay.php?f=5	benign
https://casinodemendoza.com.ar/Amileair20018/index2.php?id=1002883B100288	benign
http://web.me.com/familienholmen/hjemmesidefinalny/Forside.html/	benign
https://vedredi.hu/Yah/Yahoo!/updateY!.html	benign
http://steviaworld.com/downloader/js/cmd/	benign
http://ow.ly/pwAk30gZWEo?verify-account-disable=notice=1	benign
https://kaceauden.com/Impots/d338fca8383f1f37d191352381e94b06/	benign
http://www.locarmix.com.br/plugins/content/souchkldekl	benign
http://maannioda.com/graywbz/84c912c040936bf3e333d28993d87673-|84c912c040936bf3e333d28993d87673/	benign
https://estaegypt.org/CitiBankAcessoSeguroInternet/BRGCB/JPS/portal/Home.do.php	benign
https://files.fm/down.php?i=z6v5a2z9&n=Case+ID+923-095-834-233.html	benign
https://tiryakihaliyikama.net/temp/pdf-/pdf/	benign
https://urldefense.proofpoint.com/v2/url?u=http-3A__biokamakozmetik.com_v-3Fuid-3Derik-2Djan.ottersberg-40umusic.com-26PZUBM8LXY3S8MNRZ&d=DwMFaQ&c=o_lJda16WK5Kq4wBheKNrA&r=lnGECQZHffXEFiJUF9iYAX_YFSCcF9bNt1CqragjPjc&m=sv101HcedDbfAUizN0kPibhv-e4nleyzPUD5HGoXMwI&s=qUeRVVqdRKVpsAYOqs43FeL0Jb0JPy5yMIe3CCF0VBQ&e=	benign
http://nikoletburzynska.com/includes/cc/gade.php?https://login.srf?wa=wsignin=Xclusiv-3D|	benign
http://www.tonsing.net/plugin/wp-admin/	benign
https://bakerynepal.com/_cgi-bin/3RERZXE/12AQDWS/verification/6B5E4D3M75ND013939AM/etape1.html	benign
http://badlands.ca/gallery2	benign
http://digivoyage.com/atttttt/image.htm	benign
https://www.milaon.tk/rxc/qoqdoc/	benign
https://shop.kynoppe.com/media/mod_languages/css/get-help	benign
http://aviatraaccelerators.org/a/	benign
https://www.widoobiz.com/wp-content/themes/Widoobiz/help/assets/images/Doc/Doc/Sign/index.php	benign
http://www.krush.fm/wp-includes/pomo/entry/611311b25c470c2ead3f87987f9e7d4b/	benign

URL	Classification
https://valleynco.com/wp-includes/css/mail/login/	benign
https://knuppel.com.au/includes/onedrive/en-us/index.html	benign
https://www.leandropacheco.adv.br/index.html	benign
http://stpatricksfc.net/xs/dl/New%20folder	benign
http://aretecopiousproperties.com/bin0a00a/verification/87EC99B4C81A3AMM07DE/qes.php	benign
http://www.arteenimagen.net/golden/GDD/GDD/GDD/GDD/GD.zip/	benign
http://viddiflash.com/https/1/www.bbvanet.com.co/ecard/OTPresend.php	benign
http://highschooldropouts.info/N7277k/Adobe.pdf/adobe.php	benign
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=10&cad=rja&uact=8&ved=0ahUKEwjkp6GamKLZAhURZlAKHc41CoQQFghxMAk&url=http%3A%2F%2Fwww.soniverbags.com%2Fnews.php&usg=AOvVaw2MjQ0SSIYHGjl7kTa4CkLo	benign
hthttp://www.kostow.pl//wp-admin/img/2/index.htm	benign
https://puertolopez.gob.ec/wellsneh/indexxx.htm	benign
https://www.kardum-partneri.hr/wp-includes/css/eorro/cn/dcedb/	benign
https://gid-designferta/-/-/latest/Update.html	benign
https://grayscala.com/wp-admin/includes/user-fed/navyfederalcreditunio/nav.htm	benign
https://trainingrumahsakit.com/admin/DoCuSigN/DoCuSigN/index.php	benign
http://bradescopj.com/html/classic/index.shtm	benign
https://divyaravi.com/wp-admin/maint/index.html	benign
https://www.aurinkoharju.fi/wp-includes/js/geusg/red.php	benign
https://brazilairporttransfers.com/king/33771a4f4acb4a98be2e95f966da9beb/	benign
http://www.importarmas.com/modules/mod_feed/Alibaba.html?id=MC1IDX1gEM--h6BmklV77yUjPz71wiS-VpYO1tv3CYQmXmLbAQz8Y5KvFCNlTWFzK6mY8aa&tracelog=inquiry|view_details|100321842457&biz_type=Notifications_MC&crm_mtn_tracelog_task_id=176df315-8d37-4594-8315-fe46abf37aa5&crm_mtn_tracelog_log_id=13641815851	benign
https://t.yesware.com/tt/0b1bbdf5e8c05c6ef8f0d511bf601702d8c8e8ce/c0205025e173fa23ec343232485acd39/0e8c9124a0cd64a6d760698c3e1e390e/milehidistilling.com/how-to-make-moonshine/	benign
https://cliqmedia.com.ng/wp-admin/css/colors/light/Adobe/	benign
https://grupadellarte.pl/ofiuha/official/invoice/attached	benign
https://www.mako-store.pl/libs/cccomwellercommmm/7d1fe9e45241664682ce78547cce5475/sec2.php	benign
https://gobighub.com/.well-known/	benign
https://www.theviralscene.com/wp-adm/134a0078150ea9acfb56caba2b33b33d/	benign

URL	Classification
http://bockchrain.com/wallet/#/?gclid=EAIaIQobChMIkYji8uvx1gIVDgOGCh0q_gjBEAAYASAAEgJqefD_BwE	benign
https://www.cutterbuck.co.za/wp-admin/css/colors/ocean/1dRive	benign
https://www.gimadolciumi.com/PACK/tor/	benign
http://www.elenacoello.com/Loevo-fazer	benign
https://superiorasphalt.com/wp-includes/SimplePie/XML/Declaration/security_checkpoint/action.php	benign
https://www.aryahotel.com/core/.index.html?/fpi2vw09hjogd7jhmv/	benign
https://piscine-et-eau.fr/tmp/cm/dos/bout/particuliers/authentification/devenir/Identifiant/conseils/internet/caisses/compte/code/mutualite/Nationale/31b75/pay/ser.php	benign
https://www.lawyerdivorce.co.il/wp-inclbt0h30	benign
https://fingerpaintedblog.com/wp-admin/includes/wid-nav/navyfederal-credit-union/intel.htm	benign
http://ittechware.com/software/	benign
http://siliconebands.ca/wp-content/Gdocs/e40f487275d35fca89d8835604fece02	benign
https://www.exportgrowth.com/modules/mod_roksprocket/orangee/orange/	benign
http://premiumblue.net/modules/mod_simplefileuploadv1.3/elements/all/ii.htm	benign
http://www.sanzay.com.br/file/QDS/adm/314345/SOS9890IMPLIFICADO2017.html	benign
http://www.setarehabi.ir/AccountVerify/nsw/data/	benign
http://colmetric.com/media/jui/css/doc/drive/	benign
https://eskola.lk/kaka/attiinnddeexx.php	benign
http://www.geotalot.com/index_files/lockedfiles/e6852b48e95e60c44482a77a0b40e31c/	benign
https://kaceauden.com/Impots/119951d00a5de3f0666c120ca2c596b2/	benign
http://karasserybank.com/da9c13eff2b8731f4bdf11c95d059ecbda9c13eff2b8731f4bdf11c95d059ecb/618735a7c9209dc2ecdc8a22b793d39a618735a7c9209dc2ecdc8a22b793d39a/319c03858a4ec8cef3e03d46744fc792319c03858a4ec8cef3e03d46744fc792/Login/Login/Paypal-update/Login.php?	benign
https://tkzwhdyk.com/sqrup/squareup_com/mysquare/a10ac55cbe09b96852ec3711ca3ef250/email.php	benign
http://allmobil.com/themes/welp/customer_center/customer-IDPP00C148/2b0537b75ed9df40ce093c3e94a08d37/	benign
http://hormontant.se/wp-includes/css/picuter/www.pnc/page/verification/online/1/index.html	benign
http://bradescopj.com/html/classic/index.shtm	benign
https://www.standardintlbank.com/.redd/js.htm	benign
https://www.zipsurvey.com/Survey.aspx?suid=85067	benign
http://tienda.delivery/tommmaaaa/killer/index.html	benign
http://pulcrilin.net/tv5/images	benign

URL	Classification
https://www.puroslusitanos.com/wp-content/plugins/contact-form-7/zoominfo.com	benign
http://www.customercarenumber.co.uk/bt-com-change-password-reset-help/	benign
http://www.widoobiz.com/wp-content/themes/Widoobiz/help/assets/images/Doc/Doc/Sign/index.php	benign
https://www.blueseaalex.com/m/page/match.html#/	benign
http://www.iolagalerston.com/e-signatures/	benign
http://tienda.delivery/tommmaaaa/killer/index.html	benign
http://www.bouncewell.com/tenn//1dcad8cc5b82a7ef72f72716b220cf13	benign
http://jomjb.com/adobeaa/JSK/Adobe/adobeaa/8d81541be850c565aa97631cedc8d61f/index2.htm	benign
https://indogadget.web.id/wp-content/uploads/2014/06/ii.php	benign
https://medicalnetgroup.com/ordstrackgmluk/	benign
http://www.exmouthchapel.com/templates/exmouth2/login.php	benign
http://kirbytent.com/wp-content/plugins/capability-manager-enhanced/includes/variable.php	benign
https://secure.racine.ra.it/antispam/b.php?i=03Qak6pkF&m=e837931158ae&t=20160125&c=n	benign
http://www.forumactif.com/annuaire/jeux/world-of-warcraft/guilde-rp	benign
https://href.li/?http://www.ebay.co.uk/itm/VERY-LARGE-STONE-ACORN-FINIAL-ANTIQUE-FINml	benign
https://www.editionsadlib.com/media/news/a720351eac246fcd55c8/	benign
http://allmobil.com/themes/welp/customer_center/customer-IDPP00C148/2b0537b75ed9df40ce093c3e94a08d37/	benign
http://ashaimarineintl.com/content/spreadsheet/Excel_PO/New_PO016117.html	benign
https://www.shakinnstirred.com/support/security/paypal-team/webapps/73274a9b20c4aaf9be60c9936e8dbfe6/CreditCard.php	benign
http://www.corsiacademy.it/2010/come-si-calcola-il-valore-catastale-di-un-immobile.html	benign
http://utic.espe.edu.ec/wp-content/themes/zerif-lite/ti-prevdem/img/-/css/-/com.au/big/pond/nocharge/-/-/	benign
https://casahorus.com/hjd/Ourtime/ourtime.php	benign
http://www.assitej.pl/wpzz/roundcube/index.html?	benign
https://fullactivators.com/mac/paragon-ntfs-crack-with-keygen/	benign
http://buraphwood.com/docch/source	benign
http://moviepassage.com?subid=dropbox.cm	benign
http://solidallianzemfb.com/sollidallianze/pdf/pdf_access.php	benign
https://downuptime.net/redirect.php?url=http%3A%2F%2Fdoom-audio.de%2Fde%2Fred.html?sec=&token=null	benign
https://www.sportvsochi.ru/customer/account/	benign

URL	Classification
http://evkapen.com/in/#https://www.infomaniak.com/fr/support/payer-facturein/https://login.infomaniak.com/fr/login	benign
http://gulfuniform.com.sa/chasey/activation.htm	benign
https://grtrucking.net/update-wells/WellsLINK/	benign
http://captainjodydonewar.com/wp-content/tnroll/Bank_Of_America/verification/7CD571B73B7655204MC7/index.php?country.x=MX-Mexico	benign
https://www.gundersons.com/blog/wp-content/uploads/2016/11/Windowslive/Live/Live/Update/login2.php?https://login.live.com/public/IdentifyUser.aspx?LOB=RBGLogon	benign
https://www.colegioinfantesdelsol.cl/wp-admin/network/worth/gatehouse.capital/	benign
http://realfoodforlivingwell.com/uploads/ukk/verification.php	benign
https://msgweb.comcast.net/messaging/v2/tracking/tracking/ClickedUrl?targetId=CBFEF6AC-AFBC-4952-8558-00B3ED8FB4D8&serverId=0&targetUrl=http://customer.xfinity.com/contact-us	benign
http://www.verter.eu/modules/advancedeucompliance/tests/Unit/-/cartasi.gtwpages/e1599325e2c677d6c3f8db721a2c7dc4/main.php	benign
https://associazionepensionaticariplo.it/2-non-categorizzato/68-fondo-pensioni-cariplo	benign
https://yahoohelforusa.tumblr.com/post/170412704107/get-solution-for-yahoo-mail-error-codes#_=_	benign
https://mycasinoyellowhead.com/wp-content/plugins/.att.yahoo/01/ii.php	benign
http://www.yulynshop.co.id/wp-inv/mail/1d7bcf5d6589ebedbe4cd4fb6b3907cc	benign
https://naciwasco.org/language/overrides/wells/wells/438be3e70f/	benign
https://worldairholiday.com/wp-includes/Requests/Exception/HTTPS.php	benign
http://www.portugalkaraoke.com/atye.php	benign
https://rideforthebrand.com/rftb_images/vtu/load/viewer.php?idp=login	benign
https://lookupservclinetsrv.webs.com/nanaol.html	benign
https://ariesgrupoconstructor.com/wp-content/wireless/	benign
http://khatsaz.com/bankofamerica.com.login.account.update/1ab5b207453ce26b2c1474c0714db8da/signInScreen.go.html	benign
http://www.brambletyelodge.com.au/try/Adobe.pdf/adobe.php	benign
https://businessgroup.com.pe/wp-content/b/dropnow/dropnow	benign
https://buscasa.mx/wp-includes/ID3/wp-admin/verify.php	benign
http://sanatyemek.com.tr/onetime/	benign
http://ariben.net/wp-content/plugins/revslider/languages/temp/load/	benign
http://toenailfung.bid/Y9Un-nwV-OYTuWr07i1bSnKHyt4-PM0r-CbjlnHWZECQ	benign

URL	Classification
https://surveys.jibunu.com/Maps_0302/index.aspx?l=1&s=25477190WG&dt=20180315	benign
https://www.rsb-ol.de/libraries/redirect.php	benign
https://www.colegioinfantesdelsol.cl/wp-admin/network/worth/gatehouse.capital/	benign
https://demo.xcally.com/assets/plugins/jquery-file-upload/server/php/files/PayPalScam/9e61544a65dbc2dcc311b883a9ba3f36/?REDACTED	benign
https://vimalbearings.com/AOL/my.screenname.aol.htm	benign
https://default.salsalabs.org/T9060d681-18c8-45f8-9626-45281f8cae15/6d394b67-3d5e-473b-8726-ccce53dbdc2b	benign
https://amexopen.creditpointe.com/FileUpload/fileUploadIntroduction.htm	benign
https://treerk.ml/drpbdocs/	benign
https://files.fm/down.php?i=6mjnd8h7&n=PP-918-294-436-302.html	benign
https://abadhya.com/one/drive.live.com.signon/c85652703f710065c1255aa03a9aefae/	benign
http://abirderslife.com/dsservice/nsw/data/UntitledNotebook1.html?run=login_cmd&statuts=f17ca2c829680ada2fec9fc87bc5f6064fcb695334acc589c544ceca00d6ef7b	benign
https://www.qinetwork.com.br/como-melhorar-os-processos-da-sua-empresa-com-google-docs/	benign
https://lacausaesmexico.com/wp-admin/includes/click/index.php	benign
https://sociedadcolombianadefisica.org.co/estilo/newupload/processing.php	benign
http://www.abcmaroko.pl/includes/domit/DocuSign/	benign
https://elitecar.com.co/wp-includes/js/jquery/ui/redir.php	benign
https://toyotaclubserbia.com/Cloud/document/adobeCom/027a18ea18729799c4fcc10737acd53f/	benign
https://ekegitim.com/images/galeri/1/	benign
https://psiewdr.org/libraries/mydocument/wrong.php?email=	benign
https://www.esube-cepsubesi-teb.com/Login-Sube/Parola.do/	benign
http://sanpatricioresidencial.mx/site/	benign
http://www.lawbay.co.tz/wp-includes/js/mediaelement/rptr/index.html	benign
http://pembrokechurch.org/online/Don/http://assisttechie.com/ig-messages-recovery/	benign
https://acer.thebridgedigital.com.au/chase/customerservicealerts_/	benign
http://thegadgetisland.com/upd/Counter/index.php?email=	benign

URL	Classification
https://storage.googleapis.com/valuezon/scripts/c.html?mid=528-11608-115272534-12-8-192168212-139240074&lid=422468&i=528&c=1979&x=1121100981121141011150641121100980460991111090461121 04&t=HTM&p=2&m=11608&b=0&a=366&g=4404&l=1041161161120580470471151011101001151090971051080461101011 16&lp=/portal/528-11608-115272534-12-8-192168212-139240074/0/TR/MESSAGE	benign
http://www.babycenter.com/103_munchkin-arm-hammer-diaper-pail_10389298.bc	malicious
http://www.nexusmods.com/oblivion/users/763208/	malicious
http://www.nbcnews.com/id/10097830/	malicious
https://www.tnaflix.com/he/terms	malicious
http://www.kbb.com/car-pictures/2006-kia-sportage-suv-pictures/	malicious
https://www.vecteezy.com/free-vector/banner-pack	malicious
http://www.babycenter.com/400_how-to-manage-stress-when-not-with-my-baby_12282615_6.bc	malicious
http://www.nexusmods.com/skyrim/download/1000123559	malicious
http://tnaflix.com/robots.txt	malicious
https://serverfault.com/users/455899/user169992	malicious
http://www.cambridge.org/catalogue/catalogue.asp?isbn=9781107273214&ss=exc	malicious
http://europa.eu/pol/comm/index_el.htm	malicious
https://www.bbc.com/amharic/news-42048606	malicious
https://www.heavy-r.com/user/phenix1985	malicious
http://www.cambridge.org/cambridgelatincourse/cambridge-latin-course-4e/unit-3/	malicious
https://www.kbb.com/cadillac/cts/2005/	malicious
https://www.bbc.com/afaanoromoo/isportii-41468133	malicious
https://www.kbb.com/car-news/new-cars-arriving-in-2009/credit-score-creator-tells-how-to-increase-your-score/2000003644/	malicious
http://www.bbc.com/culture/story/20180312-the-racist-message-hidden-in-a-masterpiece?ocid=ww.social.link.twitter	malicious
https://www.kbb.com/audi/q7/2011/30t-quattro-premium-sport-utility-4d/	malicious
https://serverfault.com/a/145784/7704	malicious
http://www.babycenter.com/100_babycenter-174-reveals-profile-of-today-8217-s-millennial-mo_10391450.bc	malicious
https://www.kayak.com/Bad-Hersfeld-Hotels-Sleep-Go.256570.ksp	malicious
http://www.nbcnews.com/archive/2009/06/05/1955288.aspx	malicious
https://www.kayak.com/Athens-Hotels-Orion-Hotel.606524.ksp	malicious
https://www.vecteezy.com/free-vector/-bucket	malicious
http://www.nexusmods.com/dragonage/mods/4689/	malicious

URL	Classification
https://www.heavy-r.com/free_p**n/asian-sauna.html	malicious
https://www.usgs.gov/staff-profiles/craig-d-allen?logstash-usgs-pw%3Apalladium_root_topics=&logstash-usgs-pw%3Apalladium_root_publication_year_date=&sort=&page=4	malicious
https://serverfault.com/questions/895570/how-to-configure-samba-to-work-with-windows-10-1709	malicious
http://serverfault.com/questions/568625/server-2012-cant-open-any-file-shares	malicious
http://www.cambridge.org/aus/series/sSeries.asp?code=GAIH	malicious
https://serverfault.com/users/439008/dora	malicious
https://www.vecteezy.com/free-vector/banderas-de-marruecos	malicious
http://europa.eu/cultural-heritage/how-label-your-event-eych-2018_hu	malicious
https://www.tnaflix.com/gay-porn/?a=1&d=	malicious
https://www.cambridge.org/9781107057258	malicious
https://europa.eu/european-union/about-eu/institutions-bodies/european-investment-bank_it	malicious
https://www.vecteezy.com/free-vector/air-stairs	malicious
http://www.nexusmods.com/dragonage2/users/7018713/	malicious
https://www.vecteezy.com/free-vector/80s-vector-free	malicious
http://www.google.com/a/islagaia.pt	malicious
http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521868334&ss=fro	malicious
https://www.tnaflix.com/softcore-videos	malicious
http://europa.eu/legislation_summaries/about/index_en.htm	malicious
http://europa.eu/pol/rd/	malicious
https://www.google.com/?image_url=/gato+ojos+verdes+logo?image=352274752	malicious
https://www.heavy-r.com/p**n_videos/	malicious
http://europa.eu/geninfo/query/advSearch_el.jsp	malicious
http://serverfault.com/questions/218005/my-servers-been-hacked-emergency	malicious
http://www.nbcnews.com/business/business-news/corner-offices-patience-trump-congress-wearing-thin-n805566?cid=public-rss_20171008	malicious
http://www.bbc.com/go/mundo/nav/int/-/mundo/tiempo	malicious
https://www.kayak.com/Albertacce-Hotels.98766.hotel.ksp	malicious
https://www.kayak.com/?uuid=f46553f0-234f-11e8-ba6f-4f9f8463b0c5&vid=	malicious
https://www.heavy-r.com/free_p**n/nexus-d***o.html	malicious
http://www.bbc.com/hausa/multimedia/2013/07/000000_shirin_safe.shtml	malicious
https://www.babycenter.com/0_high-risk-moms-to-be-say-coping-with-lupus_10416160.bc	malicious

URL	Classification
http://www.bbc.com/bengali/multimedia/2013/07/130729_mk_gallery_national_geographic_photo_auction.shtml	malicious
https://www.kbb.com/buick/regal/2017/gs/	malicious
http://www.cambridge.org/asia	malicious
https://serverfault.com/users/20599/stein-%C3%85smul	malicious
https://www.kongregate.com/accounts/kilocharge12	malicious
https://www.kbb.com/bmw/3-series/1993/	malicious
http://europa.eu/newsroom/index_el.htm	malicious
http://europa.eu/legislation_summaries/glossary/competition_fi.htm	malicious
https://serverfault.com/help/badges/1/teacher?userid=84031	malicious
https://www.kayak.com/Bakewell-Hotels-The-Wheatsheaf-by-Marston-s-Inns.514715.ksp	malicious
http://www.cambridge.org/cambridgelatincourse/cambridge-latin-course-4e/unit-2/	malicious
https://www.kayak.com/Azay-le-Rideau-Hotels-Hotel-Troglododo.676562.ksp	malicious
http://serverfault.com/questions/30567/geo-dns-providers/42889	malicious
https://www.kbb.com/audi/a4/2018/premium-ownershipcosts/	malicious
https://www.usgs.gov/staff-profiles/craig-d-allen?logstash-usgs-pw%3Apalladium_root_topics=&logstash-usgs-pw%3Apalladium_root_publication_year_date=&sort=&page=2	malicious
https://www.kbb.com/car-pictures/2012-hyundai-santa-fe-suv-pictures/	malicious
https://serverfault.com/help/badges/137/quorum?userid=77231	malicious
http://europa.eu/!QH64yu	malicious
http://www.nbcnews.com/business/markets/fed-says-cut-stimulus-another-10b-stocks-take-tumble-n56886	malicious
http://www.babycenter.com/210_make-a-math-bean-game_19751_342.bc	malicious
https://www.kbb.com/car-pictures/2013-toyota-avalon-sedan-pictures/?mediaview=int&manufacturername2=toyota&modelname2=avalon	malicious
https://www.kbb.com/audi/s5/2013/premium-plus/options/?vehicleid=377739&intent=buy-new&category=convertible	malicious
https://www.vecteezy.com/free-vector/-scribble	malicious
http://www.babycenter.com/400_cramps_2385669_665.bc	malicious
https://www.vecteezy.com/free-vector/banner-de-publicidad	malicious
https://serverfault.com/questions/892590/nginx-rewrite-with-if-statement	malicious
https://www.vecteezy.com/free-vector/3d-plus-icon	malicious
https://www.google.com/a/cpanel/tria.lv	malicious
http://www.bbc.com/culture/story/20171010-the-great-writers-forgotten-by-history?ocid=ww.social.link.twitter	malicious

URL	Classification
http://serverfault.com/questions/383526/how-do-i-select-which-apache-mpm-to-use/383634	malicious
http://serverfault.com/questions/21777/archived-and-queued-windows-error-reporting	malicious
https://www.bbc.com/afaanoromoo/oduu-41040443	malicious
http://www.kbb.com/car-colors/lincoln-colors/	malicious
https://www.heavy-r.com/out.php?member=homemade-voyeur.com&url=http%3A%2F%2Fwww.homemade-voyeur.com%2Ftube%2Fvideo%2Fwaterpark-exposure-xwmYyVF9fNN.html	malicious

Table A.1: URLs in our test set which were misclassified by our classifier and the result that our classifier produced

A.2 Feature Importance

Name of module	Name of feature	Importance
Page Size	Height	0.10846
URL	IP address	0.10826
URL	Size of query	0.10012
HTML	Size of page	0.09812
URL	Hostname contains '-' symbol	0.08197
Resource	Actions single host	0.08115
Resource	Resources single host	0.06319
HTML	Number of forms	0.03748
URL	Hostname contains digit	0.02836
URL	Percent encoding	0.02203
Status	302	0.02203
URL	Trigger in path	0.01971
Resource	Relative path resources	0.01859
Resource	Same domain resources	0.01853
HTML	Scripts page ratio	0.01545
HTML	Number of password fields	0.0144
Resource	Relative path form action	0.01303
URL	Subdomain contains TLD	0.01291
URL	Has cyrillic	0.01211
HTML	Number of iframes	0.00989
Status	200	0.0088
URL	Contains '@' symbol	0.00872
Resource	Same domain form action	0.00858
URL	Changed port	0.00811
Logo Detection	Matches any logos	0.0081
Page Size	Width	0.00795

Name of module	Name of feature	Importance
URL	Path contains TLD	0.00789
Resource	External form action	0.00748
URL	Misspelled target in url	0.00697
Status	301	0.00553
URL	Target in path	0.00449
URL	Target in subdomain	0.00426
Resource	Resources actions same host	0.00405
URL	Uses HTTPS	0.00404
Resource	External resources	0.00352
URL	Contains brackets	0.00344
Logo Detection	Matches corresponding URLs	0.00294
Resource	Empty resources	0.00188
Resource	Empty form action	0.00167
Status	404	0.00164
Status	303	0.00114
Resources	Redirect	0.00076
HTML	Number tags	0.00063
URL	Shortened URL	0.00033
Status	500	0.00032
URL	Number of subdomains	0.0003
URL	Length	0.00025
Status	403	0.00024
HTML	Uses favicon	0.00014
Resources	HTTPS upgrade	2e-05
Status	401	2e-05
Status	400	0.0
Status	203	0.0
URL	Path contains '=' symbol	0.0
Status	201	0.0
Status	202	0.0
Status	204	0.0
Status	300	0.0
Status	304	0.0
Status	402	0.0
Status	501	0.0
Status	502	0.0
Status	503	0.0
Status	504	0.0
Image Resources	Contains image	0.0
Image Resources	Matches any logos	0.0
Image Resources	Matches corresponding URLs	0.0

Table A.2: List of features we have defined with associated variance obtained from our classifier