CatchPhish: A URL and Anti-Phishing Research Platform

Stephen Waddell

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

In this work, I build on the prior development of the CatchPhish Phishing Learning and Detection Tool: a browser extension designed to improve phishing detection by training users to detect malicious URLs. This prior work outlined the design of a system intended to be a novel contribution to anti-phishing research, by combining both passive indicators and active warnings into a system which actively aims to train users.

For the cumulative year of this project, the analysis aspects of the CatchPhish system have been established. This involved the development of a URL analysis server able to scalably analyse URLs with a high degree of accuracy. Through evaluations of several system designs, the final system was developed as a Microservices-based architecture deployed on Google App Engine. A heuristic algorithm was developed on top of this architecture, utilising numerous data sources to implement 46 distinct heuristics, to facilitate the system's analysis requirements. The final analysis server is found to perform well on both the system's intended deployment scenarios: accurately identifying both the high amounts of popular safe URLs typically visited by users, and also malign phishing URLs. The evaluation of the overall system illustrates the utility of CatchPhish as a versatile research platform for experimenting with different anti-phishing approaches.

Acknowledgements

First of all, I would like to thank my supervisor Kami Vaniea for all her support and encouragement throughout the entirety of this project. For the same reasons, I would also like to thank Kholoud Althobaiti for her support as a phishing expert. Both of which deserve significant acknowledgement for the considerable expertise and support they have provided across both years of this project.

I would also like to acknowledge the considerable contribution of all the participants and experts who participated in one of the many user studies and evaluations conducted throughout the entirety of this project. The efforts of all of which deserve acknowledgement.

Lastly, a particular thank you is reserved for my parents for their dedicated support, without which this work would not have been possible.

Contents

Chapter 1

Introduction

1.1 Overview

The use of phishing to exploit vulnerable users has a series of cascading impacts: with the receipt of a perceptively benign email, users can find themselves at a loss, reputationally, financially or otherwise, through the critical mistake of clicking on a single malicious link. With the harvesting of an employee's private credentials, their workplace can be exposed to infiltration. The repercussions of these attacks accumulate to have a wider economic effect - in the United Kingdom (UK) alone, phishing is expected to cost the economy as much as £280 million per year [38].

One wrong click can leave many mercilessly exposed in our data-intensive world.

Malicious URLs (Uniform Resource Locators) are one of the most common phishing delivery methods, whether through email or social media, for directing users towards malicious content. By obfuscating the real destination of a URL through a few simple manipulations, users can quickly find themselves on unknown and insecure web pages. Therefore it is vital to tackle this massive worldwide problem - the motivation behind companies such as Google [26] looking into the future of URLs themselves [73].

There are three distinct approaches to tackling phishing today. These are:

Automated Phishing Detection - using automated techniques to capture existing patterns of URL phishing signatures;

User Training - the best results being demonstrated from embedded environments, which are difficult to simulate;

Security Indicators - with contextual information being provided to encourage positive user decisions.

The limitations of these techniques are similar in how they engage users. The existing analysis techniques with most promising detection accuracies utilise machine learning, however these are rigid in their adaptability in comparison to analysing URL syntax. The imperfection of detection methods results in a performance gap users are left to bridge without the necessary training. The major issue with existing techniques is therefore with how they include users in the system design loop [16] through the information they present and when. Explainable analysis methods coupled with user focused design in real-time embedded contexts is presented as a viable way forward.

1.2 Project Specification

The purpose of this work has therefore been to build a research platform which will identify malicious phishing attacks through real-time analysis on URLs. The primary goal is to improve phishing detection rates and reduce the zero day window (the average availability time before detection) of these attacks by developing users' knowledge of phishing. The main objectives of this work consequently have been to integrate existing phishing techniques and use the resulting analysis to educate users about the URL signatures of phishing attacks.

The need for an integrated analysis approach is particularly important due to the significant zero day window associated with phishing: 32 hours and 32 minutes in the first half of 2014 [11]. The lack of standardisation within user warnings and their prevalence can lead to users ignoring key warnings when presented to them; alongside users' general ignorance of phishing techniques. In the UK, only 72% of technology users had heard of phishing as a term despite 95% of organisations saying that they train end users [56].

The focus of this paper is on the development of the cloud-based server-side analysis in the CatchPhish anti-phishing system. Built primarily for the Chrome platform [27] as a browser extension, previous work on the project involved designing the tool and implementing and evaluating the means for how users would interact with it. The requirements derived from this previous work illustrate the motivation behind focusing on the analysis objectives of the CatchPhish system in this project.

1.3 Results and Accomplishments

As part of this project, there were several significant pieces of work which are outlined below:

- completed a review of suitable cloud technologies and evaluated possible system designs whilst consulting with industry experts
- developed a microservices-based analysis infrastructure, facilitating recommended system response times [43]
- fused a mixture of data sources to implement a heuristic algorithm with 46 distinct phishing heuristics
- implemented improvements to the existing user interface, whilst adding user-considerate data gathering features

- performed a rigorous analysis of the completed system
- planned a future longitudinal study with users to derive success metrics and gather typical usage data

Of these accomplishments, there are two particularly novel aspects to this work: the integration of multiple analysis methods into a single system, with the results of this analysis used to educate users; and the combination of passive and active warnings to ensure constant threat awareness. This combination is what allows the CatchPhish system to be a research framework which can both work as a portable embedded training system whilst demonstrating reasonable phishing detection accuracies - by placing users firmly at the core of secure system design.

1.4 Report Structure

This report describes the most critical elements of this research. The remaining nine chapters are structured as follows:

Chapter 2: discusses the previous work on the system and illustrates the goals for this iteration of the project.

Chapter 3: presents an overview of phishing including the common approaches to tackling it with an analysis of existing anti-phishing systems.

Chapter 4: outlines the work done to design the architecture of the system's analysis server.

Chapter 5: provides an overview of, and key motivations behind, the server architecture and the anti-phishing heuristic algorithm.

Chapter 6: focuses on the implementation of specific improvements on the previous years work, required for future studies.

Chapter 7: discusses the steps taken to improve the system before deployment - appropriate threshold selection and performance improvements.

Chapter 8: critically evaluates the analysis accuracy and UI improvements, whilst outlining preparations for the longitudinal study.

Chapter 9: interprets the study results and presents a discussion on the ethics and privacy considerations of the system.

Chapter 10: concludes this work and provides an overview of further work beyond this project.

Chapter 2

Previous Work

The original motivation behind this project was the lack of tools which both prevent users from visiting malicious URLs and effectively explaining the reasoning behind this decision making. This is why previous work on this project aimed to solve the question:

How might we develop a phishing learning and detection tool that will protect from, and inform users about, malicious URLs?

This chapter outlines the previous work done to answer that question and critically looks at areas for improvement.

2.1 Requirements Gathering and Design

The basis of this project was to develop on the work of the Faheem slackbot [9] by implementing a system which would be able to present the user with explanatory URL information in a more interventionist and integrated context: utilising work in anti-phishing UI design [8] and influenced by Volkamer et al.[65] in approach.

A browser extension was selected to implement this system due to its ubiquity: allowing for all web requests arising from the clicking of a URL within, or without, the user's browser to be filtered by this system. The Chrome browser was therefore chosen because it has largest market share of any currently used browser (62.5% globally) [66]: intended to give the tool as wide a reach as possible. Developing for this browser is also well documented [24, 25].

The tool has been designed for, and tested with, users with above average technical skills as the main userbase for the tool. Over-confidence in their knowledge of phishing [67], the focus on this userbase in established work [8] and existing research on how technical knowledge disseminates itself among the wider population were also motivating reasons.[1]

[1]Users with above average technical skills are most likely to be asked about and share their knowledge with users with less adequate technical skills [68, 49, 53].

2.1.1 Interviews

Interviews with the selected userbase were conducted to gather requirements for two purposes: to understand how much knowledge of phishing this userbase have and how they would like to interact with the tool, with subsequent reference to the user interface options available within a Chrome Extension. The participants were further separated according to computer security experience. The results of 17 participants were analysed as part of these interviews, with each interview lasting around one hour.

2.1.1.1 Phishing Knowledge and Design Results

The results of the interviews demonstrated participants were largely able to detect phishing emails and point out relevant indicators (the average success rate being 97.91%). Participants largely knew the purpose of URLs, but had less of an ability to adequately read them. The participants particularly struggled with detecting more difficult aspects of URL reading, such as detecting non-ASCII characters or understanding shortened links. This lead to a conclusion that users do not understand URLs sufficiently to be able to detect malicious indicators, a conclusion supported by existing research [5].

Participants had a variety of ideas of how and what should be presented in the Chrome extension, with participants splitting into distinct groups based on the level of intervention they wished for in the tool.

Based on these results, the UI elements were chosen by selecting the most popular features and integrating these into a compatible combination - whilst allowing options for different user configurations.

2.1.2 Algorithm and UI Design

Paper prototyping was carried out to develop these requirements into a usable UI. The design was largely based on the work of Althobaiti et al. [8] with minor functional adaptions.

The design of the analysis algorithm was a key component for the tool to be explainable. The algorithm type was chosen to be a heuristic algorithm for two reasons.[2] A lack of labelled data meant it would have been needlessly challenging to develop a machine learning algorithm which could have improved on the accuracy rates of Google's.[3] Particularly with the knowledge that the tool would be able to benefit from API access to Google's safe browsing data [29]. Secondly, there was a need to process the URLs in such a way that would provide information for the front-end to display. In machine learning approaches, it is difficult to understand how individual features in data map to the end result - which limits the explainability of these approaches.

[2]A heuristic algorithm is a type of algorithm used to combine multiple metrics which can give an approximation of a solution.

[3]A classifier based on Random Forrest which achieves a 90% accuracy on noisy data [17, 69]

The algorithm was designed to aggregate the results of a set of defined heuristics: each producing a status value (or a severity level) and combined to form an overall count of the status types by matching these values against a set of established thresholds. These status values ordered by increasing severity are: ‘none’, ‘possible’ and ‘known’. These heuristic status values are aggregated to classify the URL into the following classes in order of safety: ‘safe’, ‘warn’ and ‘alert’. The algorithm was also designed to analyse a bespoke set of safety metrics to evaluate the safety of a URL. This was a design choice intended to reduce the algorithm’s future false negative rate. Whilst a framework for analysing a hypothetical set of safety metrics was considered in the prior year, the design and development of the safety algorithm and metrics was a primary focus of this year’s work as described in Section 5.2.

One issue with employing heuristic algorithms is the difficultly of selecting appropriate heuristics which allow the algorithm to have a high accuracy. This is difficult for phishing in particular as it requires a significant amount of research into a literature dense research field. For this reason a report into potential heuristics was developed in the prior year by analysing work on existing phishing research (see Appendix A).

Both the UI and the algorithm design were evaluated by an expert in phishing and URLs. The expert was positive about the design of the tool and suggested several particular improvements, drawing on examples such as the Netcraft extension interface [42]. They felt the algorithmic approach was constructive and they provided further resources to increase the breadth of heuristics.

2.1.3 Requirements

As a result of the requirements gathering and design stage, the high level design goals of this system were defined as follows:

- **Requirement 1**: Comprehensiveness; classifying every URL a user encounters - whether it is present in a web page, a page URL itself or any URL which would be processed during a web request in the browser.
- **Requirement 2**: Automatic Malicious Blocking; preventing the user loading data from any resource where the URL has been classified as malicious, without explicit user consent.
- **Requirement 3**: Understandable; presenting URL information to users clearly, both at appropriate points of intervention and on user request.
- **Requirement 4**: Best Practice Engineering; that the final system maximises efficiency and accuracy whilst providing configurability and a positive user experience.

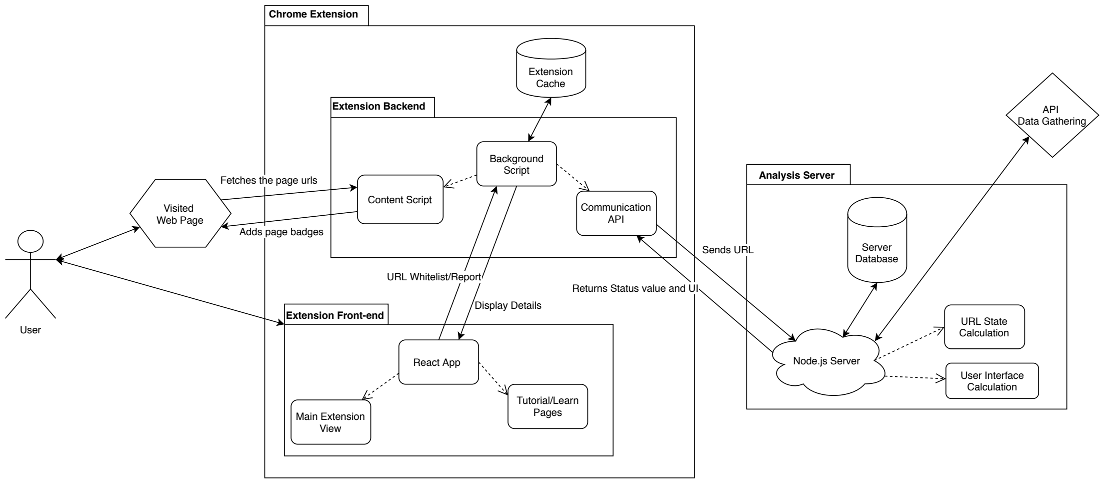


Figure 2.1: System Component Overview

2.2 CatchPhish Implementation

The CatchPhish system leverages existing web technologies to derive information about URLs. The system works by extracting all the URLs from each page the user arrives on. These are forwarded to a decision-making server which parses and handles the analysis of the URL. The resulting system provides extensive information about each URL in a user-friendly manner.

Figure 2.1 demonstrates the three major components of the system:

- **Chrome Extension Infrastructure** - handles the overall application logic, the extension's interactive elements and page alterations.
- **Front-end** - responsible for the main user URL analysis display, along with tutorial pages, built using a React app incorporating Material Design.[4]
- **Analysis Server** - responsible for the URL calculations, built using the Node.js framework.[5]

The Chrome Extension APIs [25, 23] were used to implement many of the UI elements of the tool that were chosen as a result of the interviews:

- a context menu entry (on right click) to access link details
- a badge over the plugin icon to display the overall site information
- link annotations with a status for each link on a page

The extension sends all user web requests to the server for analysis, with malicious requests blocked and the information presented to the user in an embedded training simulation - with no content from a page loaded before a request is blocked.

[4]React is a Javascript library for creating user interfaces [52]. Material Design is a design language created by Google and incorporated into their products which focuses on using responsive animation, padding, and depth effects to make the interactions clear to the users [18].

[5]A flexible runtime environment for running Javascript outside of the brower [44].

React was used to implement users' desire for a popup with a detailed information breakdown, along with the tutorial and settings pages, with the intervention page design being based on the existing Google site warning [28]. Figure 2.2 displays the main popup summary page of the extension.

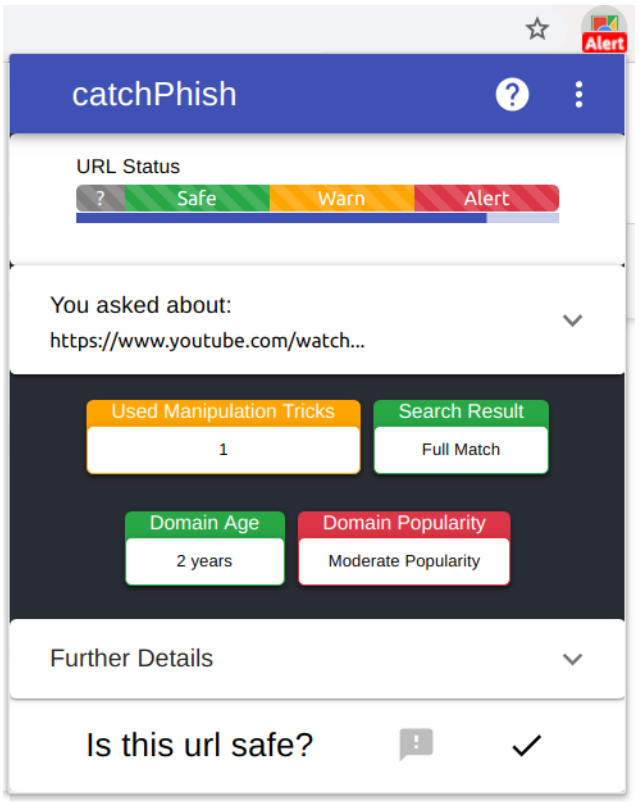


Figure 2.2: Popup Summary

The detailed information breakdown represented in the Further Details section of the UI, required the dynamic generation of user interface statements. These user statements were required to adapt their explanations of the heuristics to match the heuristics' identified severity levels. This was found to be a key feature required to educate users in prior research [8]. This was therefore a further focus of this year's work as outlined in Chapter 5.

The Analysis Server was mocked up using Node.js, as a rough design of a processing framework. The key takeaway from this implementation was the interface for interacting with the Frontend. As discussed, the Frontend was designed to be dynamically generated based on the heuristics it receives: specifically the Further Details section. The server implementation consisted of around four implemented heuristics and some experimental features such as the ability to unshorten links implemented in Javascript.

Security issues of the tool were also considered, with efforts made to protect user data on the extension side (by hashing URLs present in its cache). Further improvements to the security of the system were highlighted as issues that needed to be addressed.

2.3 Evaluation and Further Work

A triangulated evaluation approach was taken to evaluate the tool using a closed demo environment (a server hosting a clone of a popular site).

A survey containing System Usability Scale (SUS) questions was conducted at a project open day.[6] The survey had 43 responses receiving an overall score of 81.3 - within the SUS range of very good.

Think Alouds were used to get a more detailed understanding of specific issues users might have with CatchPhish.[7] The eight participants were largely positive and users on the whole had few reported issues with the tool itself, with a few minor issues that caused confusion; 87.5% participants indicated they were *Fairly likely* to use the tool.

For the final evaluation two experts in phishing were consulted for an expert evaluation of the project. Both experts overall impressions were highly constructive (being particularly impressed with the amount and quality of work achieved in the limited timeframe). Beyond this, the evaluation highlighted some areas of the system design that could be improved such as the system security and specific approaches used to implement the heuristics.

2.4 Resulting Priorities

There were three particular priorities resulting from the previous work on this project.

The need to develop URL analysis components was a priority: involving the implementation of each of the heuristics outlined in the algorithm design report (Appendix A). The feedback from the expert evaluation suggested improvements in the heuristics by using alternative language libraries - requiring multi-language support in the design. The system also lacked robustness and its efficiency required thorough evaluation.

The user interface also required some improvements, closely related to the server decision-making heuristics. For the original UI design to be fully implemented, each heuristic required an explanation that could be adapted to suit its final state: each of which needed to be dynamically generated based on the contents of the URL and highlighted appropriately in the UI.

Whilst the system was able to meet three of the design requirements, its ability to meet the requirement of Best Practice Engineering or its educational merit required further evaluation. Therefore a longitudinal study was marked as a key priority: requiring the addition of data analytics to be built into the system.

[6]The System Usability Scale consists of a ten item questionnaire with five response options for respondents: from *Strongly Disagree* to *Strongly Agree*.

[7]With Think Alouds, participants' emotional responses allow researchers to build an understanding of how users actually use an application. Participants work through a set of tasks whilst talking aloud: voicing their thoughts and actions [30].

2.5 Summary

This chapter gives an overview of previous work done to implement the anti-phishing system outlined in this report. Key points highlighted within include the decision to build a Chrome browser extension focusing on above-average technical users: with system requirements being derived from a literature review, interviews with 17 participants and consultation with experts in the field.

The designed URL analysis algorithm and an overview of the infrastructure of the system is also presented; with results on how the system was evaluated: through a survey of 43 participants, eight Think Alouds and two expert evaluations. The conclusive result of the prior evaluation finding that the previous system was usable. This chapter concludes by highlighting the multiple aspects of the system that required further development.

Chapter 3

Related Work

This chapter provides an overview of relevant research in the area of phishing for the purposes of understanding the novelty of the completed work.

3.1 Background Information

Phishing is a means of using deception to acquire private and confidential details from users.

The variants of phishing range in sophistication [63]. Deceptive Phishing is the most common type of phishing scam and it hinges on how closely the attack email resembles a legitimate company's official correspondence, largely distributed on-mass to multiple recipients in a blanket approach. More sophisticated approaches such as Spear Phishing use more personalised attacks, using details such as recipient's name, position and company to trick the recipient into believing the attacker has a personal connection with them [14]. These more sophisticated methods are an increasingly popular choice [41, 56] over conventional phishing because of their high success rate [34, 60].

Emails are often regarded as the main delivery vehicle of phishing attacks. With 54.6% of all email being spam, it can be hard for users to differentiate between malicious and legitimate emails. Malicious emails can be caught by the users' email clients or spam filter, but this has a varying effectiveness [72, 15, 59]. Phishing attacks can, however, be spread through any online communications means that allow for the use of links or attachments [71].

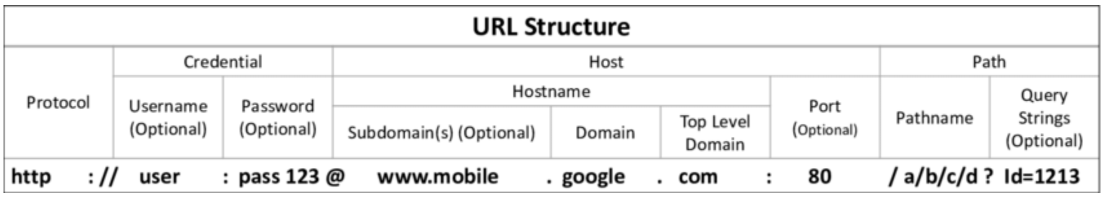
3.1.1 URLs and Manipulation Attacks

A URL is a specification mechanism for pointing to and indicating how to retrieve resources on a computer network.[1] URLs are utilised extensively in the Internet for

[1] URLs are often colloquially called a web addresses or links.

referencing web pages (http/https), file transfer (ftp), email (mailto), database access (JDBC) and several other purposes.

Their use for multiple purposes is allowed by their flexible structure, outlined in Figure 3.1.



URL Structure								
	Credential		Host				Path	
			Hostname					
Protocol	Username (Optional)	Password (Optional)	Subdomain(s) (Optional)	Domain	Top Level Domain	Port (Optional)	Pathname	Query Strings (Optional)
http ://	user	: pass 123 @	www.mobile	. google	. com :	80	/ a/b/c/d ?	Id=1213

Figure 3.1: URL structure example [9]

URL adaptability means that expertise is required to truly understand them. Internet users have been shown to lack this expertise, having difficulty with reading URLs [9, 8, 11, 58]. Even after having extensive training in the subject users can still fail to notice visually deceptive manipulations [19]. This leaves internet users susceptible to attack without persistent or accessible URL knowledge.

Attacks which involve the manipulation of URLs are known as URL manipulation attacks, and can be part of both phishing and other attacks which employ URLs. A Quarterly Threat Report by proofpoint highlighted: "the pendulum of malware delivery mechanisms in email continued to swing towards URLs; malicious URLs outnumbered attachments ... by over 370%."[50].

3.1.2 Common Indicators of Phishing Attacks

There are multiple indicators of phishing attacks, but these depend on the type of phishing delivery vehicle used. An indicator can be used as a gauge or measure of whether a phishing attack has occurred, and examples of these are discussed in the subsequent sections.

3.1.2.1 Email Specific Indicators

There are a range of email specific indicators depending on how sophisticated the attack is. Lower quality phishing emails also include indicators that people are more familiar with: poor spelling and grammar are common indicators. In contrast, more sophisticated spear phishing attack can employ tactics such as email spoofing [55], so that the message appears to have originated from a non-malicious source.

3.1.2.2 Common URL Manipulation Tricks

Users can be tricked through URL manipulations intended to confuse the user into clicking on a malicious link. One common set of URL manipulations is known as *"mangle"*: where a brand or company name has letter substitution, misspelling or non-ASCII characters that appear to be similar to their English counterparts [35] . The use of non-ASCII characters means this is very difficult to identify at a brief glance and users do not typically spend a while focusing on URLs in the page [65].

3.1.2.3 Domain Indicators

The domain name of a site embedded in a site's URL (Figure 3.1) can be used to view metadata which can provide indicators of phishing.[2] One of theses indicators is the age of the domain. Domain databases such as WHOIS [70] can be queried to get the creation date of the website. This can then be evaluated against known thresholds for typical phishing sites.

3.1.2.4 Page Indicators

Page features use information about web pages which are calculated using reputation ranking services. One page indicator is the relative popularity of a website which can be determined by looking at information by using Alexa Top Sites [6] to get a rank of the most popular domains [37, 75]. Popular sites tend not to be phishing sites since users do not tend to revisit a phishing site. In this sense, the highly popular sites in a list such as Alexa's Top Sites can be used as a whitelist; whereas the most unpopular sites from this list (or absence from the list) can be taken as being a malicious site.

3.1.3 Advanced Attack Patterns

Beyond indicators which can be evaluated based on syntax analysis or metadata thresholding, there are advanced attack patterns which are more difficult to detect. Covert redirect or cloaking [61] is a trick used to make the visible destination of a phishing URL suggest a reliable destination whilst redirecting users. This can be difficult to detect as it may only occur for certain users. One method to achieve this is by overriding browser mouseovers,[3] which can lead to users visiting malicious sites without their prior awareness [13].

3.2 Research Approaches

Each phishing research approach analyses phishing indicators, but they are distinguished by both how they derive their analysis and how they apply them: whether

[2]A domain name is a label which identifies a network domain - a unique group of computers that form part of a central administration or authority. These domain names are managed by the Domain Name System (DNS).

[3]A mouseover is a feature of many desktop email clients and web browsers which display a hypertext link's target URL in the status bar while a mouse hovers over it.

that is automated detection or user training.

3.2.1 Automated Phishing Detection

Automated phishing detection is an approach favoured by large organisations and companies. This approach employs machine learning techniques to analyse URLs to discern if they are malicious.

These techniques can be used to populate stored blacklists.[4] These databases are often maintained independently by companies, with some publicly available for querying. One of the significant benefits of using blacklists is the reduction in computation time needed to process whether each URL is malicious. Blacklists tend to be highly reliable when analysing the maliciousness of their present URLs, but since they are not complete lists, they cannot be fully relied on. This requires URLs outwith blacklists to be analysed.

The main drawback of these approaches are the false warnings they can create which decrease user confidence in the effectiveness of the prevention systems [57]. This is why automated approaches such as machine learning are not effective as a singular indicator of phishing attacks.

3.2.2 Security Indicators

The basis of automated security indicators are to feed contextual information about security practices back to the user so they can make educated decisions. Common examples of this approach include informative mouseovers or the highlighting of important information for the users [65].

A further example of automated security indicators is given in the URL report research [8]. This research presents a detailed breakdown of how a URL's different components can be presented to a user. The report allows for extensive contextual information about each URL element, whilst highlighting the most important aspects of a URL to a user. The report was created with the use of focus groups in order to be an effective and useful presentational tool. It breaks down the information about URLs into three primary sections: summary, URL manipulation tricks and facts about the URL. For each fact, there is a clear explanation of why it is significant. This is presented alongside an example of the URL component itself, to make the connection between the fact and URL component clear to the user. The theoretical UI outlined in this report formed a basis in the design of the CatchPhish UI.

[4]Blacklists are extensive databases of phishing links which are known to be dangerous [48].

3.2.3 Training Users

Existing research [16] highlights the need for the capability of users to be included in security system design where they cannot be removed, rather than being treated as an enemy [2] of an effective system. This is why it is necessary to adequately train users to be useful components of a security system [46] and make use of their significant potential: humans are often more capable at spotting unique and unknown patterns than machines.

Training users is therefore a critical approach of tackling phishing and it can improve the ability of users to make effective decisions when faced with phishing attacks. There are multiple approaches to training users and many different tools designed to help with this. Facets of ideal training approaches are: engaging the user, presenting relevant information of the subject and having clear learning objectives.

Training through the use of educational games is an approach intended to engage the user by presenting information in fun environment. This, however, is subject to the quality of the game. One common difficulty with this approach, and training in general, is the user's ability to retain information taught to them after some arbitrary period of time. After learning new information users tend to show increased knowledge and skills on the subject immediately afterwards, but this is shown to deteriorate over time. This is a concern with computer security applications specifically as it is essential that users continue to have the knowledge and awareness that allows them to adapt to threatening issues as they appear. In phishing literature an example of this can be found in the NoPhish app [11]. This is an app intended to teach users about phishing using informative pieces of information. As a result of the longitudinal study included in the paper, they found that users were more successful after the teaching, but their success rates dropped five months later when tested on the same material again.

Embedded training has been found to have particularly successful training results. The basis of this approach is to teach users in the moment they make an error, about why that error has occurred. This helps to improve user retention by creating memory anchor points which help users recall the learned information when faced with similar situations.

A particular problem with embedded training is that it is difficult to deploy in real life situations. In corporate environments it is challenging to set-up a scenario for embedded training where the employees do not know it is occurring. The training therefore fails to train users appropriately as they adapt their behaviour to the training environment. It is also difficult to create the environment in which these kinds of training can occur without a high amount of intervention into typical user routines.

3.3 Existing Solutions

There are several existing anti-phishing tools, the majority of which are incorporated into general security-focused software such as anti-virus software. These comprehensive software packages tend to focus on providing the user with a sparse amount of high-level information, limiting their educational impact. They are designed to keep users engaged by highlighting functionality whilst limiting users ability to grow out of their need for the software - these tools are often paid for and the average user tends to have limited knowledge of computer security in general.

Established security extensions such as Netcraft's are an example of existing bespoke anti-phishing tools. Netcraft's browser extension in particular is designed to present information about the quality of the site to indicate to users it is phishing or not and act as passive indicators, with further details only triggered by user concern.

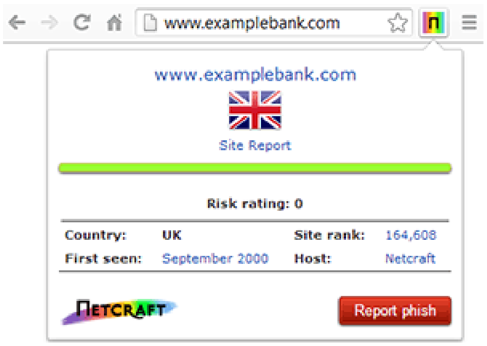


Figure 3.2: Netcraft extension interface [42]

These security toolbars are often not designed to convey information clearly to users, and essential heuristics such as the popularity of the site can be ignored by users as a result. This can be due to a lack of contextual information explaining why this information should be important for user's decision making. Wu et al. [74] discuss the usability of these security toolbars in their paper, with Netcraft's security toolbar (Figure 3.2) an included example. The paper itself raises future design principles for anti-phishing tools, drawn from research into the usability limitations of the example security toolbars themselves. The design principles suggest "active interruption like the pop-up warnings [are] far more effective than the passive warnings" but they "should always appear at the right time with the right warning message" in order to ensure user trust in the system.

An anti-phishing tool built into a web browser itself is the Google site warning which employ active intervention. These are presented to prevent users from visiting malicious sites, by using the information calculated as part of Google's automated phishing detection. This warning can sometimes be thrown when the browser processes links that are malicious, to indicate that the subsequent site is malicious. This is useful as an intermediary for preventing users from visiting and being immediately affected by the malicious site. However, the usefulness of this security feature is limited by users trust in these warnings, regardless of the accuracy of Google's aforementioned classifier work.

Other existing tools have attempted to improve on user warnings using active intervention, such as in the work of Yang et al. [75] who tested a chrome extension to warn users of malicious phishing. Their field experiment suggested an active intervention approach could be incorporated into a future anti-phishing tool, as long as the user was presented with an understanding of why they had been intervened. This complements the inclusion of embedded training in such a tool by displaying the URL analysis to the user at appropriate points of intervention.

3.4 Summary

This chapter underlines what phishing is, and provides an explanation of the different vehicles used to deliver phishing attacks. It discusses the indicators used to identify these attacks and the depth of technical knowledge that is often required in order to understand what constitutes a phishing attack.

Further, this chapter provides a depth of information on the current techniques to tackle the problem of phishing attacks. Research approaches such as automated phishing detection are analysed whilst the effectiveness of user training and automated security indicators are discussed. These research approaches are compared with the anti-phishing solutions presented by existing tools.

Chapter 4

Designing the Analysis Server

There were several considerations that were deliberated when designing the Analysis Server architecture. This chapter outlines these considerations along with a comparison of viable designs.

4.1 System Features and Requirements

The system was required to have the following core features:

- An event-driven URL analysis pipeline
- Storage of URL analysis along with user usage statistics
- Automated data analysis for researchers
- Scheduled updates of preprocessed data for the URL analysis functionality
- Adaptable creation of UI explanations for each heuristic tailored to severity

The system was also ideally required to allow the later integration of additional features as might suit future research goals. For instance, utilisation of machine learning approaches for secondary data analysis.

The requirements of the system were defined to be:

- **Processing Speed**: A URL analysis response time between 0.1 and 1 seconds.[1]
- **Data Storage**: Ability to store and access system data without a significant impact on system performance, alongside allowing access to longer term study results. Data structures for quick lookups on natural language data are also required.
- **Allows for Async Preprocessing**: The periodic population of database white- and black- lists should have a minimal performance impact on the core system.

[1]This is an important range for ensuring user engagement [43].

- **Language Flexibility**: To make use of the best libraries for processing URLs across various languages.
- **Authorisation**: Be able to prioritise requests from different sources (e.g. Browser Extension over Web Sites), being able to distinguish between these.
- **Data Security**: The security of user data is an utmost priority - secure communication between all stakeholders and the server, along with an authorisation system for database access.

Further optional requirements included the ability to store prior versions of the URL analysis by collection timestamp, along with being able to process heuristics and their data sources concurrently to improve response times.

4.2 Server Designs

The system had to incorporate several key features whilst balancing the outlined requirements. These were weighed across the multiple Server design proposals.

Due to the volume of users planned for future user studies, being able to deploy the system on an external cloud platform was a priority. The Infrastructure-as-a-Service (IaaS) providers were filtered by those able to provide long-term support for future user studies. This, therefore, limited server designs to platforms provided by AWS and Google who were able to provide this support.

4.2.1 Dedicated Server Design

The Dedicated Server design utilised a largely monolithic architecture, incorporating most system features into a processing application deployed on a dedicated server. This design was most similar to the existing work on the mock-server and allowed for significant re-use of resources.

The Dedicated Server stack utilises the Express web framework [2] to facilitate a simple REST API. Node URL libraries, along with additional language libraries and Web APIs are utilised for the URL preprocessing. PostgreSQL[3] is the Relational Database Management System (RDMS) utilised to store the URL analysis and TULIPs lab resources. Database authentication is managed using PostgreSQL user accounts, with authorisation of Chrome extension users utilising OAuth2[4]. Data preprocessing and web scraping is implemented in Python, with further scripts employed to automate data analytics. The system is intended to be hosted on a dedicated server such as an EC2[5] instance.

[2]https://expressjs.com

[3]https://www.postgresql.org/

[4]https://oauth.net/2/

[5]https://aws.amazon.com/ec2/

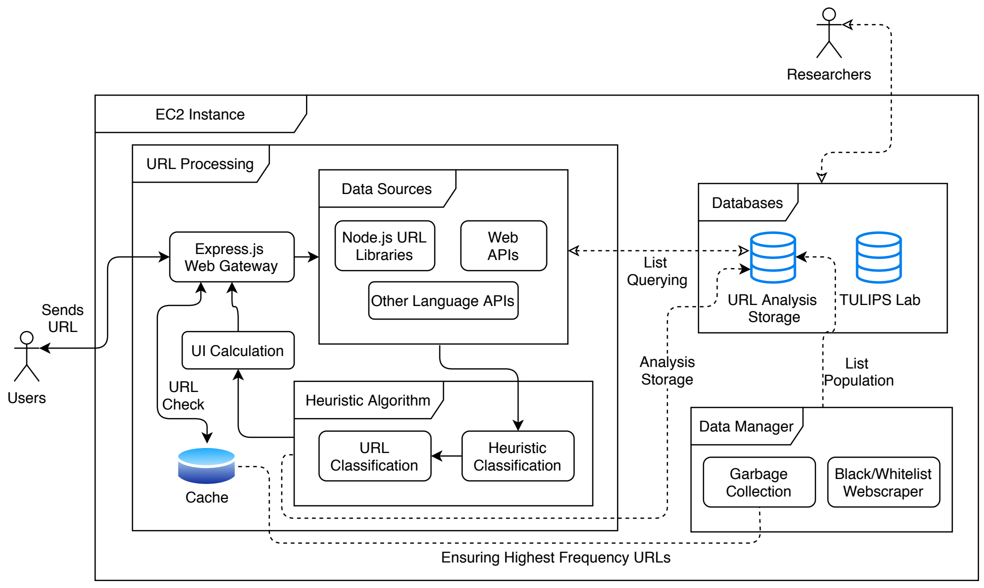


Figure 4.1: Dedicated Server design

A simulation was conducted using this architecture in order to evaluate both the robustness of the system and the amount of user requests the system might be expected to process per unique user. Understanding the amount of user requests per day was a requirement for evaluating the level of processing power required by the system. To facilitate this study, the mock-server was updated: a database was added to store user requests and a domain was purchased to facilitate TLS with LetsEncrypt [21].

Count	Mean	Std Dev	Min	25.00%	50.00%	75.00%	Max
24364	4,060	1,248	2,790	3,166	3,807	4,619	6,116

Figure 4.2: Key statistics on the unique URLs processed by server during the user trial. These results are only drawn from the six day period the system was fully functional to ensure their reliability.

As I fall into the category of users this tool is intended for, I evaluated the tool using data produced from my own browsing habits (with the trial being conducted between the 27/10/2019 and the 13/11/2019). My user data provided an estimate of how many URLs a typical user would require the server to analyse. The analysis of this data indicated that 4060 unique URLs per day would need to be processed on average, as illustrated by the results in Table 4.2.[6] After discussions with industry experts (outlined in Section 4.2.2), they suggested the amount of URLs per day highlighted the need for

[6]There is limited use for these results beyond their use as an estimation due to possibility of temporal or seasonal variance. A larger pool of users across a longer time frame would be required to produce more meaningful results.

a more scalable architecture to provide the level of performance required by multiple users.

The simulation also highlighted the limited robustness of the monolithic architecture: of the 15 days the monolithic architecture was trialled, only 6 days were unaffected by server faults. This trial provided an example of the difficulties which are caused by handling external user data. The results of this trial underlined a need for alternative methods, such as containerisation and orchestration, to improve the system robustness.

4.2.2 AWS Serverless Design

The AWS Serverless design was intended to implement the same system functionality. This design splits the integrated system features of the Dedicated Server design into interconnected subsystems utilising separate AWS products. The primary benefit being added system scalablity, a feature of serverless architectures, which helps to improve performance.

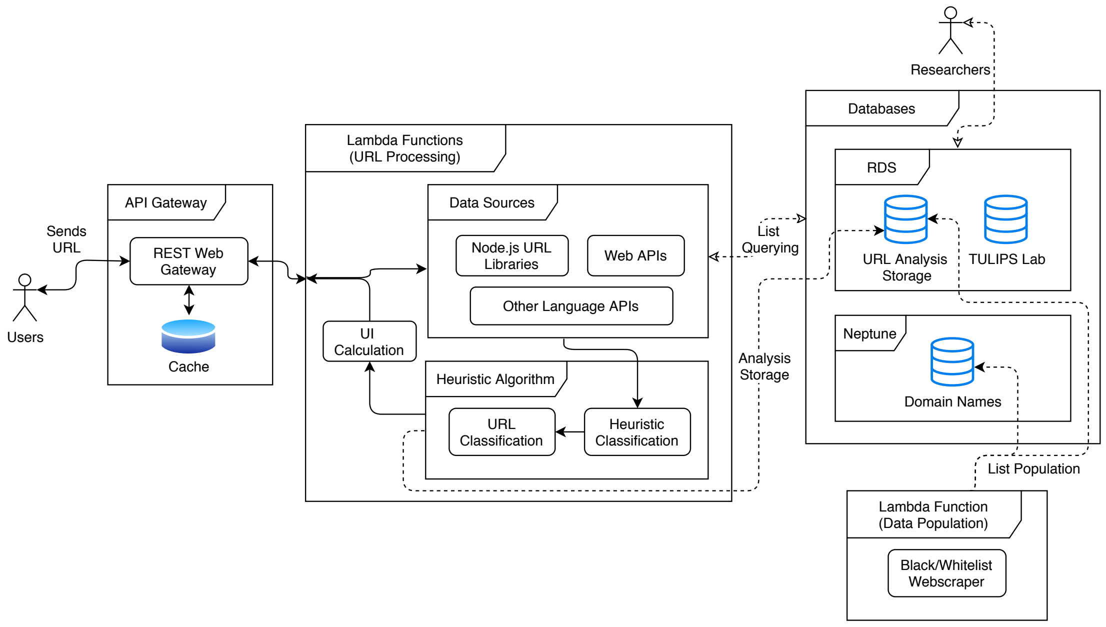


Figure 4.3: AWS Serverless design

API Gateway[7] would replace the Express.js Web Gateway. This gateway service manages traffic to existing back-end systems and also includes result caching. HTTPS endpoints and SSL certificates can also be setup with the API Gateway[8].

Lambda functions[9][10] implement the data sources, event-driven database access and

[7]https://aws.amazon.com/api-gateway/

[8]https://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started-client-side-ssl-authentication.html#configure-api

[9]https://docs.aws.amazon.com/lambda/latest/dg/lambda-functions.html

[10]AWS Lamda[11] is a serverless compute service which allows some functionally sized pieces of code to be run based on event-based triggers.

heuristic algorithm processing. Lambda functions have been designed to implement application logic, by triggering other Lambda functions if they meet the logic requirements (e.g. processing an additional URL on detection of a shortened link). The URL Processing application is split into an application-like structure with multiple functions triggered by distinct events. AWS Lambda Layers[12] provide additional language support by allowing Lambda Functions to pull in additional libraries written in different languages and/or run-time environments, and share code between functions.

A separate application is designed to implement the scheduled data updating (black- and white- list preprocessing). Lambda Functions can be triggered by inbuilt CRON jobs, which would allow for the necessary independent scheduling[13].

For data storage, AWS's RDS server is used with PostgreSQL support allowing the same database structure, and data separation, as the dedicated server. Automated and encrypted database backups are a feature of RDS database.[14] Neptune[15] is a Graph database service planned to hold a memory-intensive graph-data structure (for the implementation of complicated heuristic), since Lambda functions have no continuous memory. Incorporating Neptune was intended to reduce the need for memory management using a separate subsystem as may be needed in the Dedicated Server system design.

As part of the design of this system, AWS Solutions Architects were consulted on the final design. Several consultations on the system design occurred alongside funding discussions. Both the Dedicated Server and AWS Serverless system designs were sent for evaluation as part of a written up design proposal (see Appendix B). The feedback on this system design was overwhelmingly positive, with a suggestion made to incorporate AWS Step Functions to coordinate all of the Lambda Functions using a simple graphical interface.

4.2.3 Google App Engine Design

Google App Engine is a fully managed serverless application platform. Automatic scaling is implemented for web applications, by horizontally scaling applications to met demand. It includes the levels of sandbox service abstraction and orchestration [39] offered by the use of Docker and Kubernetes. Google App Engine supports multiple runtimes and frameworks including Node.js and Python.

A Microservices architecture is strongly supported by App Engine. Microservices architectures split system features into small independent (Micro-)services which communicate through messaging [20]. As an architecture which facilitates the use of orchestration technologies for system scalability [1, 20] - Microservice architectures are well placed to facilitate the data processing demands required by this system. Using

[12]https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html

[13]https://docs.aws.amazon.com/lambda/latest/dg/tutorial-scheduled-events-schedule-expressions.html

[14]https://aws.amazon.com/rds/details/backup/

[15]https://aws.amazon.com/neptune/

this architecture allows distinct services to be used to implement individual features of the analysis server design. For instance, individual services for UI support and analysing the URLs. The App Engine platform allows these services to be scaled to meet demand and be developed with the language best suited to facilitate each of their requirements.

The design therefore includes separate applications for both UI calculation and URL processing, separating system features into distinct applications. More information on this design can be found in Chapter 5.

4.3 Comparison of Designs

All of the proposed system designs implement the core system features, with some variation on how each handle the requirements.

Both the AWS Serverless and Google App Engine system designs remove the need to consider machine performance by providing a horizontally scaling system architecture which abstracts individual machine performance. Compared to the Dedicated Server design, these reduce the need for future development required to scale the system, whilst limiting the need to implement performance management. This allows development to focus on the general research approach such as the performance of individual heuristics. It also simplifies the wider deployment of the tool in the future and potentially increases robustness of the system. Whilst monolithic applications have been shown to have greater throughput as standalone applications than Microservices architectures [4], this does not include the performance benefits provided by their greater scalabilty [1]. Therefore these benefits alongside the greater robustness and flexibility the serverless system designs are more suitable for the system architecture than the Dedicated Server design.

However, the AWS Serverless system design is critically limited in its portability and complexity. The design and implementation of multiple disparate lambda functions requires the complex function coordination solely provided by AWS, significantly reducing the portability of the codebase for future developers: composed of a group of disparate functions rather than interconnected code joined by app frameworks.

These limitations are handled well within the Google App Engine design. Distinct system features are able to be separated by their aggregate purpose as applications, rather than as coordinated functions, improving the portability. The system also incorporates inbuilt layers platform security like the AWS services, but more closely integrated with the applications themselves. Dedicated platform applications such as Memorystore[16] are also highly useful additions, over the caching presented by the AWS' API Gateway, allowing the cache to more closely tailored to the needs of particular services.

Therefore, the chosen system design is the Google App Engine design due to its greater long-term portability and useful dedicated applications.

[16]https://cloud.google.com/memorystore

4.4 Summary

In this chapter, several architectures were outlined for the system analysis server. These designs were contrasted against the outlined system features and requirements in order to highlight the choice of final design. The final design chosen was based on Google App Engine - due to its beneficial platform level performance benefits and its ease of development, for this system, in contrast to other serverless architectures.

Chapter 5

Server Implementation

The Analysis Server is an integral part of the CatchPhish system necessary for the implementation of real-time phishing detection. This chapter outlines the implementation of the Server along with the phishing analysis algorithm.

5.1 Server Architecture

An overview of the final server architecture is outlined in Figure 5.1.

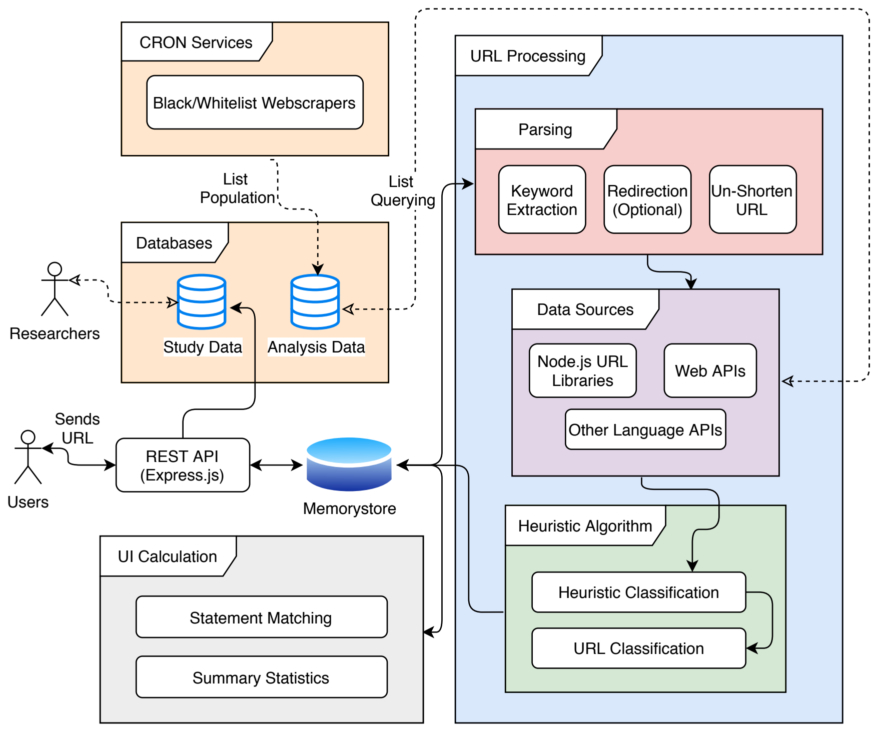


Figure 5.1: Analysis Server architecture (utilising the Google App Engine platform)

The current stack utilises a Microservices architecture to split distinct features into small independent services. It utlises the Express.js web framework[1] to facilitate a simple REST API. Python URL libraries, Web APIs and preprocessed URL/phishing data is utilised as part of the dediciated URL Processing app. PostgreSQL[2] is the RDMS utilised to store the user data and URL analysis, with app data separately stored. Database authentication is managed using PostgreSQL user accounts and platform level features. Data preprocessing and web scraping is implemented in Python as distinct CRON services, with further scripts employed to automate data analytics. The system is serverless with apps/services scaled to suit demand, all of which is facilitated by the Google App Engine platform.

5.1.1 Core Services/Applications

- **REST API**: A system entry point with a HTTPS endpoint receiving all data in POST requests. It interfaces with the other applications and triggers them based on the received request. It records both the user request data and analysis data in the Study Database.
- **URL Processing**: A dedicated python app for processing and classifying URLs. The app utilises a layered architecture to encapsulate data in independent layers.
- **UI Calculation**: Python app for processing user-facing explanations for each heuristic, calculated based on its level. The statements produced are displayed on the Browser Extension UI. This application only adds its UI results to requests from Browser Extensions.

5.1.2 URL Processing Layers

- **Parsing**: The parsing layer unshortens any URL and splits the URL into its component features, extracting the keywords. Redirection checking, and associated heuristics, are an additional optional feature due to their unconstrainable system time usage.
- **Data Sources**: Analyses multiple data sources of three distinct types: web data, lexical data and database list data (containing both contextual and known white/blacklists data).
- **Heuristic Algorithm**: Heuristics process the data and utilise vulnerability thresholds to produce an individual classification for each heuristic. All heuristics are processed as an independent layer which are aggregated as part of the classification layer to produce an overall URL classification.

[1]https://expressjs.com
[2]https://www.postgresql.org/

5.1.3 Caching and Databases

- **Memorystore**: Intended to decrease access times by storing common processing requests for a limited size selection of highly frequent URLs.
- **Databases**: Two isolated PostgreSQL databases are contained in the same management system. The App Data database contains the black- and white- lists, along with contextual information useful for URL keyword parsing. The Study Data database contains the processed URLs, user extension usage history and stores the collected URLs of each unique page processed by the app.

5.1.4 Data Management and Security

- **CRON Services (Async scheduling)**: Scheduled web scraping tasks populate the App Data database, updated independently of the core system to ensure data freshness. The times for updating each list are set according to the data sources recommended update schedule.
- **Automated Data Analysis**: Scripts to download and process the latest usage and system analysis information.
- **System Security**: Built-in platform level features are enabled such as Google Cloud Security Scanner and the Secret Manager API.[3]

5.1.5 Motivation for Key Architectural Decisions

There were several key architectural decisions made to facilitate the system requirements (Section 4.1).

Principal of these, was the decision to utilise a layered architectural pattern [54] for the URL Processing application. This was a design decision made to ensure a separation of concerns between independent application components. Each layer has a specific role and responsibility within the application, which is dependent on multiple sources of information in the previous layer. The benefit of this approach is to make it easier to maintain the application due to its well-defined component interfaces. An alternative layering approach of tying data sources with their dependent heuristics was considered to be significantly more challenging to maintain.

Choosing the right service boundaries is a common issue in Microservices architectures [54, 33]. Defining an application boundary between the UI Calculation and URL Processing apps was a choice made to reduce the data processing involved in the processing app. This increases the portability of the processing app - allowing it to be easily extracted from the system and re-purposed. A drawback of this approach is

[3]Google Cloud Security Scanner performs automated penetration testing on App Engine apps, whilst the Secret Manger API stores sensitive data such as API keys, passwords, and certificates.

maintainability: each time a heuristic is added to the URL Processing app its UI explanations have to be added to a separate application. On balance, this maintainability drawback is outweighed by the applications' increased portability.

Application independence can also allow the architecture to utilise the languages and frameworks best suited to implement system features. For instance, the Express.js Node framework is highly suited to be the system entry point due to its inherent async capabilities [62]. In contrast to the Dedicated Server design (outlined in Chapter 4), by making the system entry point independent of the processing application, user requests are better distributed around the system. The use of Memorystore also complements this choice of feature independence. The functional requirement for recording user data is implemented by logging all system requests at the entry point. Attempts to cache requests to the Express server would lead to data loss, by preventing event logging. The placement of Memorystore instead facilitates data collection whilst minimising the need to reprocess URLs, by caching the processed analysis from the URL Processing and UI Calculation apps.

Microservices architectures are also known to increase system security if they are complemented by coherent data management plans like Database per Service [40, 76]. If attackers are able to exploit vulnerabilities in one application, they only have access to the data associated with that. This pattern has been implemented as part of this architecture to ensure data sourced from more insecure sources is distinct from user data. To complement this, different levels of access control have been implemented to ensure only appropriate users can get access to each type of data. Each application also ensures its input is sanitised to reduce the risk of compromise.

5.2 Algorithm Heuristics

Central to the system is the heuristic algorithm implemented in the URL Processing app. The algorithm works by processing multiple distinct heuristics, assigning each a severity level and aggregating the level occurrences. URLs are classified based on how the aggregated counts compare with established thresholds. Numerous data sources have had to be consulted and processed to facilitate each heuristic. Since it is critical there are as few false negative results as possible, the safety metrics are evaluated according to a separate bespoke algorithm.

5.2.1 Data Sources

Figure 5.2 illustrates a comprehensive list of data sources utilised by the system. There were three distinct types of data sources:

- **Database Data**: long-term multi-use data sources such as URL and domain black/whitelists along with contextualising data such as Forbes Top Companies.
- **Lexical Data**: information specific to the URLs themselves and added to by additional libraries.

- **Web Data**: data gathered through external web requests comprising multiple sources.

The data sources were selected to fulfil the requirements of the outlined heuristics. During the process of selection, the sources (a mix of APIs, websites, and downloadable files) were evaluated to determine their reliability and ensure the consistency of their interfaces.

Whilst lexical and web request data have to be retrieved for each URL request, database data can be shared between individual requests. To increase the efficiency of the system this was therefore gathered asynchronously to the analysis pipeline and stored in the App Data database.

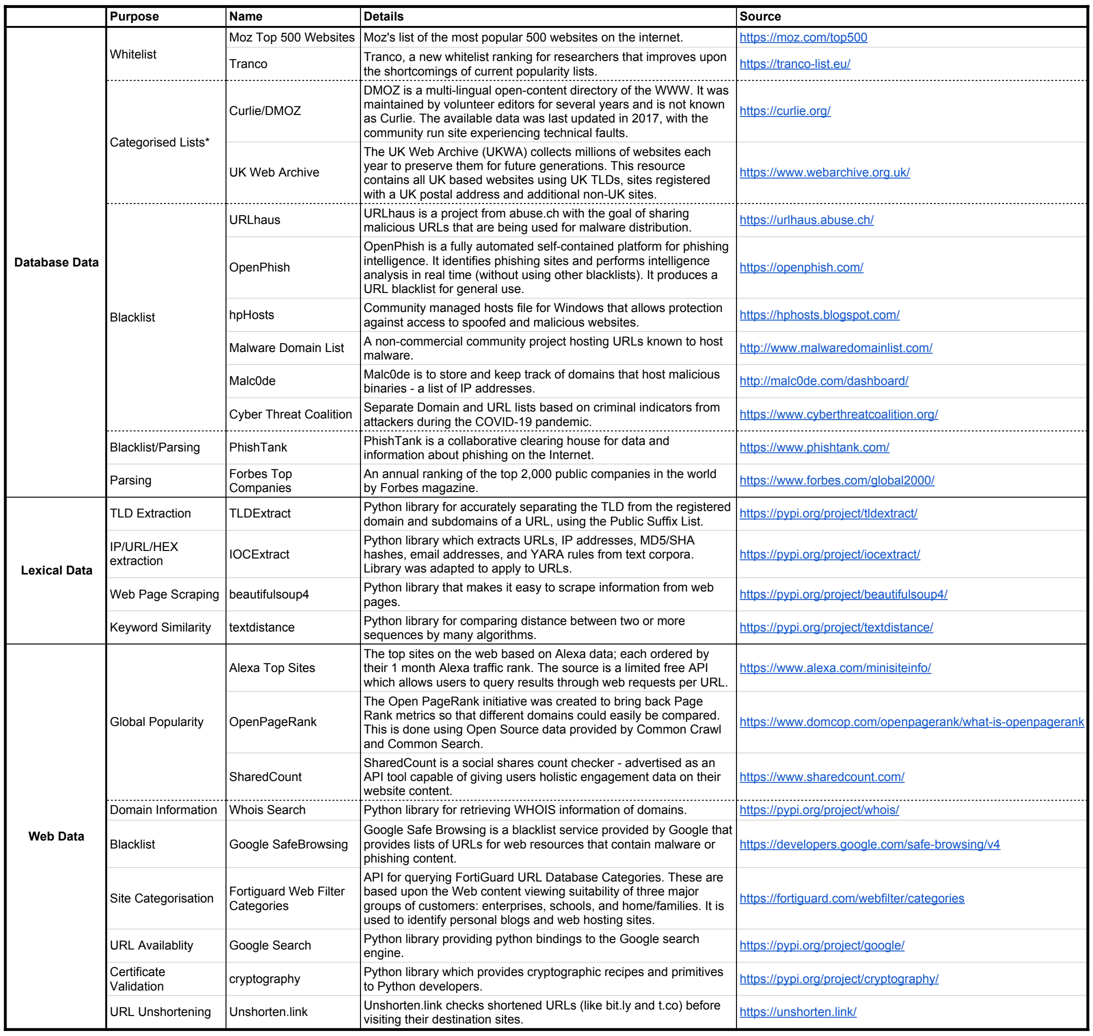
Multiple blacklists were utilised as part of the system to increase the known phishing result coverage. Using multiple white and blacklists to make use of non-overlapping URLs is encouraged by research [10]. There is some level of overlap between the chosen blacklists, with lists significantly larger than others (1.6 million in Google Safe Browsing). Foremost of the whitelists is the Tranco list [36]. Tranco is a domain data set aggregating multiple whitelists into a list which is intended to significantly improve on the “similarity, stability, representativeness, responsiveness and benignness” of existing whitelists - the validity of which are questioned. They further highlight the vulnerabilty of these whitelists to adversarial manipulation and have built in counter measures into the Tranco list.

Whilst utilising both SharedCount and the Forbes Top Companies sites as data sources, security vulnerabilities were found in their sites. Both sites exposed their private API keys to the wider public, exposing them to information disclosure attacks. Both sources were subsequently informed of these vulnerabilities and encouraged to rectify them.

5.2.2 Heuristics

Much of the research behind these heuristics are drawn from existing work [7]. The referenced work was used as the key source for the phishing heuristic proposal outlined in Appendix A, which was supplemented by further research. The heuristics are largely drawn from key features identified by machine learning alongside other anti-phishing work.

Overall, there are 46 heuristics implemented as part of the analysis algorithm. The high amount of heuristics was a necessity to develop an algorithm which suits this particular task [51, 7]. Each of these required the extraction of relevant data, the fusion of data sources and research into appropriate thresholds. A single heuristic can have multiple severity levels depending on whether the underlying data is continuous or discrete. For each severity level of each heuristic, an explanatory statement was added to contextualise the heuristic relative to its identified severity level. These statements were incorporated into the UI Calculation app. Figure 5.4 illustrates the full collection of these heuristics and their corresponding statements.



	Purpose	Name	Details	Source
Database Data	Whitelist	Moz Top 500 Websites	Moz's list of the most popular 500 websites on the internet.	https://moz.com/top500
		Tranco	Tranco, a new whitelist ranking for researchers that improves upon the shortcomings of current popularity lists.	https://tranco-list.eu/
	Categorised Lists*	Curlie/DMOZ	DMOZ is a multi-lingual open-content directory of the WWW. It was maintained by volunteer editors for several years and is not known as Curlie. The available data was last updated in 2017, with the community run site experiencing technical faults.	https://curlie.org/
		UK Web Archive	The UK Web Archive (UKWA) collects millions of websites each year to preserve them for future generations. This resource contains all UK based websites using UK TLDs, sites registered with a UK postal address and additional non-UK sites.	https://www.webarchive.org.uk/
	Blacklist	URLhaus	URLhaus is a project from abuse.ch with the goal of sharing malicious URLs that are being used for malware distribution.	https://urlhaus.abuse.ch/
		OpenPhish	OpenPhish is a fully automated self-contained platform for phishing intelligence. It identifies phishing sites and performs intelligence analysis in real time (without using other blacklists). It produces a URL blacklist for general use.	https://openphish.com/
		hpHosts	Community managed hosts file for Windows that allows protection against access to spoofed and malicious websites.	https://hphosts.blogspot.com/
		Malware Domain List	A non-commercial community project hosting URLs known to host malware.	http://www.malwaredomainlist.com/
		Malc0de	Malc0de is to store and keep track of domains that host malicious binaries - a list of IP addresses.	http://malc0de.com/dashboard/
		Cyber Threat Coalition	Separate Domain and URL lists based on criminal indicators from attackers during the COVID-19 pandemic.	https://www.cyberthreatcoalition.org/
	Blacklist/Parsing	PhishTank	PhishTank is a collaborative clearing house for data and information about phishing on the Internet.	https://www.phishtank.com/
	Parsing	Forbes Top Companies	An annual ranking of the top 2,000 public companies in the world by Forbes magazine.	https://www.forbes.com/global2000/
Lexical Data	TLD Extraction	TLDExtract	Python library for accurately separating the TLD from the registered domain and subdomains of a URL, using the Public Suffix List.	https://pypi.org/project/tldextract/
	IP/URL/HEX extraction	IOCExtract	Python library which extracts URLs, IP addresses, MD5/SHA hashes, email addresses, and YARA rules from text corpora. Library was adapted to apply to URLs.	https://pypi.org/project/iocextract/
	Web Page Scraping	beautifulsoup4	Python library that makes it easy to scrape information from web pages.	https://pypi.org/project/beautifulsoup4/
	Keyword Similarity	textdistance	Python library for comparing distance between two or more sequences by many algorithms.	https://pypi.org/project/textdistance/
Web Data	Global Popularity	Alexa Top Sites	The top sites on the web based on Alexa data; each ordered by their 1 month Alexa traffic rank. The source is a limited free API which allows users to query results through web requests per URL.	https://www.alexa.com/minisiteinfo/
		OpenPageRank	The Open PageRank initiative was created to bring back Page Rank metrics so that different domains could easily be compared. This is done using Open Source data provided by Common Crawl and Common Search.	https://www.domcop.com/openpagerank/what-is-openpagerank
		SharedCount	SharedCount is a social shares count checker - advertised as an API tool capable of giving users holistic engagement data on their website content.	https://www.sharedcount.com/
	Domain Information	Whois Search	Python library for retrieving WHOIS information of domains.	https://pypi.org/project/whois/
	Blacklist	Google SafeBrowsing	Google Safe Browsing is a blacklist service provided by Google that provides lists of URLs for web resources that contain malware or phishing content.	https://developers.google.com/safe-browsing/v4
	Site Categorisation	Fortiguard Web Filter Categories	API for querying FortiGuard URL Database Categories. These are based upon the Web content viewing suitability of three major groups of customers: enterprises, schools, and home/families. It is used to identify personal blogs and web hosting sites.	https://fortiguard.com/webfilter/categories
	URL Availablity	Google Search	Python library providing python bindings to the Google search engine.	https://pypi.org/project/google/
	Certificate Validation	cryptography	Python library which provides cryptographic recipes and primitives to Python developers.	https://pypi.org/project/cryptography/
	URL Unshortening	Unshorten.link	Unshorten.link checks shortened URLs (like bit.ly and t.co) before visiting their destination sites.	https://unshorten.link/

Figure 5.2: List of all external data sources and key libraries used in the Analysis Server. These were collected through a thorough exploration of data source options and supplemented with key Web Data sources, such as Open PageRank, through consultation with a PhD student. *The categorised list datasets were added to the system after the system trial discussed in Section 7.3.

	Name	Safety Priority	Data Used in UI	UI Explanations	UI Evidence Focus
Safety Metrics	Presence in Whitelist	Primary	Whitelist Name, Domain	The URL's domain is present on a list of known safe site domains.	Whitelist Name
	Presence in Categorised Data*	Primary	Categorised List Name, Domain	The URL's domain is present on a list of safe categorised domains.	Categorised List Name
	Extended Validation	Primary	Domain	The owner of this site has purchased an Extended Validation Certificate which verifies their identify with trusted third parties.	Domain
	High Global Popularity	Secondary	Alexa Traffic Rank	This URL's domain has a high presence in global traffic rankings. Sites that are more visited tend to be safer.	Alexa Traffic Rank
	High Sites Linking In	Secondary	Sites Linking In	This URL's domain suggests a high number of sites link to this sites with this domain, an indicator of safety.	Sites Linking In
	High Page Rank	Secondary	Page Rank Decimal	The Page Rank of this URL is high. This search engine metric suggests the URL is popular, which can indicate URL safety.	Page Rank Decimal
	Fortiguard Safe Domain*	Secondary Support	Domain Category, Domain	This URL's domain has been marked with a safe category by the Fortiguard Web Filtering service.	Domain Category
	Hostname in Search Results	Secondary Support	Match Status, Matched Link	The hostname of this site has been matched in the Top 10 Google search results: [Match Status].	Matched Link
	High Social Reputation	Secondary Support	Facebook Share Count, Pinterest Count	This URL has a high share count on Facebook and/or Pinterest. Sharing popularity can be a moderate indicator of safety.	Facebook Share Count, Pinterest Count

Figure 5.3: List of all safety metrics implemented. Further details on the selection of safety metric priority levels can be found in Section 5.2.2.2 and user statements can be found in Chapter 7. *These heuristics were only added to the system after the trial discussed in Section 7.3.

	Name	Severity Level	Data Used in UI	UI Explanations	UI Evidence Focus
URL Manipulation Tricks	Too Many Subdomains	Known	Subdomain Count, Combined Subdomain	Most organisations use zero to two subdomains but this uses at least 15 subdomains at [Subdomain Count] subdomains.	Combined Subdomain
		Possible		Most organisations use zero to two subdomains but this uses at least three subdomains at [Subdomain Count] subdomains.	
	Amount of Digits in Hostname	Known	Digit Count, Hostname	This URL has over 5 digits in its hostname - it has [Digit Count] digits. A high amount of digits in the URL hostname may indicate randomness and is a malicious indicator of phishing URLs (30%).	Hostname
		Possible		This URL has digits in its hostname - it has [Digit Count] digits. Digits in the URL hostname may indicate randomness and are common in the host of phishing URLs.	
	Number of HTTP and HTTPS	Known	Protocol Count, Most Common Protocol	There is at least the presence of two HTTP or HTTPS in the URL, with [Protocol Count] present.	Most Common Protocol
		Possible		Two HTTP or HTTPS protocols are present in the URL, with [Protocol Count] present.	
	Typosquatting Popular Domains	Known	Similar Domain, Domain	Typosquatting is practice of creating similar domains with slight differences in spelling to highly popular domains. This URL's domain is highly similar to [Similar Domain].	Domain
	UTF-8 Encoding Substitution	Known	Encoded Character, Location In URL	The [Encoded Character] is in this URL's hostname. UTF8 encoding can also be used to produce identical-looking characters from different languages and alphabets.	Location In URL
	Occurrence of Mislead	Known	Company Name, Location In URL	The name of the popular company "[Company Name]" is embedded somewhere in the URL but not the destination.	Location In URL
	Check for I.P. Address	Known	Occurrence Count, I.P. Address	An I.P. Address has been used in the URL. There are [Occurrence Count] occurances of I.P. Addresses in this URL.	IP Address
	Check for Hex Encoded URL	Known	Hostname	The hostname of this URL has been hex encoded.	Hostname
	Check for a Dotless IP in the URL	Known	Location In URL, I.P. Address	A dotless IP is a 32-bit number which resolves into its equivalent dotted IP format. This noted as [I.P. Address] in this URL.	Location In URL
	Non Standard Port	Known	Port Used, Location In URL	The port used is [Port Used]. Phishers use different port numbers to escape the detection, other than standard ports such as: 80 and 8080.	Location In URL
	TLD Out of Position	Known	Popular TLD, Subdomain	A highly popular top level domain [Popular TLD] had been found in this URL's subdomain.	Subdomain
	Different Top Level Domain	Known	Whitelist Name, Original TLD, Most Popular TLD, Domain	The URL has a different top level domain to a highly popular site. This URL's TLD is [Original TLD] compared to [Most Popular TLD] of the domain found in the [Whitelist Name] whitelist.	Domain
	Has Abused TLD	Known	Abused TLD, Location In URL	A top level domain commonly associated with phishing attacks, "[Abused TLD]", has been used in this URL.	Domain
	Unusually Long URL Hostname	Possible	Hostname Length, Hostname	The URL has over 70 characters in its hostname at [Hostname Length] characters, which is a malicious indicator.	Hostname
	Suspicious Characters in the URL	Possible	Character, Location In URL	The "[Suspicious Character]" is in this URL. This is a character commonly used in phishing URLs.	Location In URL
	Top Phishing Target	Possible	Target Name, Popularity of Target, Location In URL	This URL includes the name of a top phishing target from the last 30 days - [Target Name] which the [Popularity of Target] most targeted.	Location In URL
	Embedded URL in Query String	Possible	Embedded URL Count, Query String	There is at least one URL in the query string of the URL - [Embedded URL Count] embedded URLs.	Query String
	Amount of Hypens in URL	Possible	Hyphen Count, Location In URL	There are two or more hypens in this URL with [Hyphen Count] hyphens. This is a malicious phishing indicator.	Location In URL
	Unusual Top Level Domain	Possible	TLD, Domain	This URL has an unusual top level domain "[TLD]" as it is not in the most commonly used.	Domain
	Is Shortened URL	Possible	Original URL	This was previously a shortened URL which was expanded for analysis.	Original URL
	No Encryption Used	Possible	N/A	N/A	N/A

	Name	Severity Level	Data Used in UI	UI Explanations	UI Evidence Focus
Domain	Domain Age	Known	Domain, Domain Age	55% of phishing sites have been found to be less than a day old.	Domain Age
		Possible		95% of phishing sites tend to be less than six months old.	
	Domain Blacklisted	Known	Blacklist Name, Domain	The domain is present on a list of known malicious domains.	Blacklist Name
	URL Blacklisted	Known	Blacklist Name, Hostname	This URL is present on a list of known malicious URLs.	Blacklist Name
	Google Safe Browsing Blacklisted	Known	Blacklist Reason	This URL is listed in Google Safe Browsing which has a collection of over 1.6 milliion blacklisted URLs.	Blacklist Reason
	Suspicious Domain Category	Known	Domain Category, Domain	This URL has a suspicious domain category according to the FortiGuard web filter.	Domain Category
	No Whois Domain Match	Possible	Domain	There is no match for this domain in the public WHOIS records; no data on who responsible for this domain name or underlying IP address.	Domain
	Registrant Name Hidden	Possible	Registrar, Domain	The owner of the site has masked their identify behind their domain registrar: [Registrar].	Registrar
	Domain Indicates Hosting Services	None	Domain Category, Domain	This URL has a suspicious domain category. "Personal Websites and Blogs" and "Web Hosting" are both known to more commonly host malicious URLs.	Domain Category
Page	Number of Redirections	Known	Amount Redirected	This URL has over six URLs for which it has been redirected to which is a malicious indicator.	Amount Redirected
		Possible		This URL has been redirected between 3-6 times which is a sign it is possibly malicious.	
	Number of Shortened Redirections	Known	Amount Of Shortened URLs	This URL has over one shortened redirections which is malicious indicator.	Amount Of Shortened URLs
		Possible		80% of phishing tweets have been shown to have at least one shortened redirection.	
	Low Global Popularity	Known	Alexa Traffic Rank	There is no data on this URL's domain or it has an extremely low presence in global traffic rankings. Low popularity sites tend to be malicious.	Alexa Traffic Rank
		Possible		This domain has an fairly low presence in global traffic rankings. Low popularity sites tend to be malicious.	
	Low Page Rank	Known	Page Rank Decimal	There is no data on this URL or it has an extremely low page rank. Low popularity sites tend to be malicious.	Page Rank Decimal
		Possible		This URL has an fairly low page rank. Low popularity sites tend to be malicious.	
	No Presence in Search Results	Known	Match Status, Matched Link	There is no presence of the domain in the Top 10 Google search results: [Match Status].	Matched Link
		Possible		There is no presence of the hostname in the Top 10 Google search results: [Match Status].	
	Low Sites Linking In	Known	Sites Linking In	There is no data on the number of sites linking to pages with this URL's domain. Low popularity sites tend to be malicious.	Sites Linking In
	Number of Blacklisted Redirections	Known	Blacklisted Count, Example List Name, Example URL, Example URL Count	[Blacklisted Count] blacklisted URLs including a URL present [Example URL Count] times in blacklists such as [Example List Name].	Example URL
	Low Social Reputation	Possible	Facebook Share Count	This URL has a low share count on Facebook.	Facebook Share Count

Figure 5.4: List of all heuristics implemented as part of the analysis algorithm. Further detail on the selection of the severity levels and user statements can be found in Chapter 7.

5.2.2.1 Developing the Heuristics

The heuristics implementation involved tackling an extensive range of challenges. Whilst multiple heuristics were added on the 34 outlined in the heuristic proposal, not all of those proposed heuristics were able to be implemented due to the bounded time constraints resulting from the system response time requirement. Specifically, evaluating the digit replacement of letters - whether key words in a URL have had their characters replaced with digits: such as ‘o’ with ‘0’. There are four particular heuristics which help to illustrate the depth of work done in this area: Typosquatting Popular Domains, Different TLD and the keyword analysis heuristics: Occurrence of Mislead and Top Phishing Target.

The Typosquatting Popular Domain heuristic was challenging to implement due to the response time requirements. Typosquatting involves the creation of domains which have small spelling deviations from highly popular ones. This manipulation attack can be used, as an example, to target users who incorrectly type a web address into their browser. To implement this heuristic with a reasonable response time, only the domains included in the system’s whitelists are compared with the processed URL domain using a similarity metric. This helped to bound the computation time in comparison to checking with spelling deviations in larger corpora. The final implementation of this heuristic was established by using database SQL queries to calculate the levenshtein distance between the relevant word tokens. This implementation shaved a minute off of the URL Processing app response time, in comparison to the comparative non-database implementation.

The Different TLD heuristic was also challenging to implement in a given response time. This heuristic identifies where a URL has the same domain as a popular site but a different top level domain. To implement this, a processed URL’s TLD and primary domain were extracted. These were then compared with the primary domains in the whitelists, gauging for both a primary domain and TLD match. It was found to be highly inefficient to reprocess the TLDs of each of the whitelisted URLs for each request. Therefore, the data processing and database tables, were each updated to apply TLD extraction to the URL domains when populating the database lists. This was one approach to improve the heuristic performance.

The use of Forbes Top Companies was a creative choice to make use of a data source typically unused for phishing analysis. This data source was used to evaluate the Occurrence of Mislead. This heuristic is focused on checking if a popular company name is embedded somewhere in the URL outside of the destination. The Forbes Top Companies list was therefore a useful list of potential companies targeted for phishing attacks.

In a similar manner, further data insights were able to be derived from PhishTank beyond its use as a blacklist. With each URL the Phishtank API provides the target of blacklisted URLs. This was, therefore, repurposed to provide the most popular phishing targets for the last 30 days. This data was incorporated into the Top Phishing Target heuristic to evaluate if a URL keyword refers to a company which is a top target.

The implementation of these heuristics was complemented by consultations with a phishing PhD student. They provided suggestions for implementation approaches for specific heuristics, particularly the Extended Validation heuristic and the set of redirection heuristics.

5.2.2.2 Safety Metrics

There are a limited number of metrics which can verifiably indicate a URL is safe. In the bespoke safety algorithm, there are two ways that a URL can be classified as safe: if a URL has no heuristics with a 'known' or 'possible' severity level or it satisfies certain safety metric conditions.

Of the safety metric conditions, there is a top level of metrics which can give an independent definitive result: either Presence in a Whitelist or Extended Validation. With a positive result in either of these categories, the URL can be marked as safe with a high degree of confidence.

There is second set of conditions focused on analysing the popularity of the URL. These involve combining multiple popularity metrics to establish a high degree of confidence. Where URLs have a high global popularity (derived from the traffic ranking on Alexa Top Sites) combined with a High Page Rank and a high number of Sites Linking In, the URL can be marked as safe.[4] If one of these these metrics is not satisfied, the Social Reputation and Domain in Search Result heuristics can each act as supporting indicators which increase the likelihood of URL safety for indefinite cases.

This second set of metrics is based on the phishing research into the links between URL popularity and phishing. Popularity measures, such as a site's traffic rank, are a strong indicator of safety, as users do not regularly return to sites known to be malicious. Page Rank, for example, works by measuring a pages inbound and outbound links - a metric previously established by major search engines to measure site popularity. Safety indicators are also derived from social media sites such as Facebook and Twitter which regularly include links to external sites shared between its users. Multiple shares suggests a site may be safe, based on typical user sharing patterns. However, known phishing patterns limit the strength of indicators such as social reputation as social media sites can be potential vehicles of phishing attacks.

5.2.2.3 Redirection Heuristics

The redirection heuristics are largely included in the system for their ability to provide further analysis options for researchers on static datasets; rather than for incorporation in the real-time system. This is largely because it was not possible to utilise these heuristics and maintain a bounded system response time. This is due to the unknown nature of how many redirections might occur for each URL and a requirement to follow the full redirection chain for the heuristics to both function and have accurate information presented to users.

[4]Conversely, the low popularity of URLs may be an indication they are possibly malicious [7].

Therefore, whilst the redirection heuristics are included in this work they are explicitly not evaluated as part of the included studies which are focused on real-time contexts. The decision to discount these optional heuristics from the system evaluation therefore also allows the response time of the system to be measured for a typical use case. This has limited impact on the overall approach as heuristics have also been shown in prior research to have a negligible impact on the overall classification performance in comparison to that of other heuristics [7].

5.3 Summary

This chapter outlines the Analysis Server implementation: its basis as a Microservices architecture deployed on Google App Engine with distinct apps for URL Processing, UI Calculation and database population. The chapter also details the implementation of the analysis algorithm, providing a comprehensive list of the data sources utilised for URL classification. The 46 implemented heuristics integral to the algorithm are also outlined, with the implementation of key heuristics such as Typosquatting discussed. The bespoke algorithm for evaluating URL safety is also presented.

Chapter 6

Browser Extension Improvements

To facilitate its future use as a research platform, several modifications had to be made to the existing CatchPhish system.

This chapter focuses on these modifications, with a particular focus on UI improvements. The deployment of the tool on the Chrome store is also briefly discussed.

6.1 Additional Features

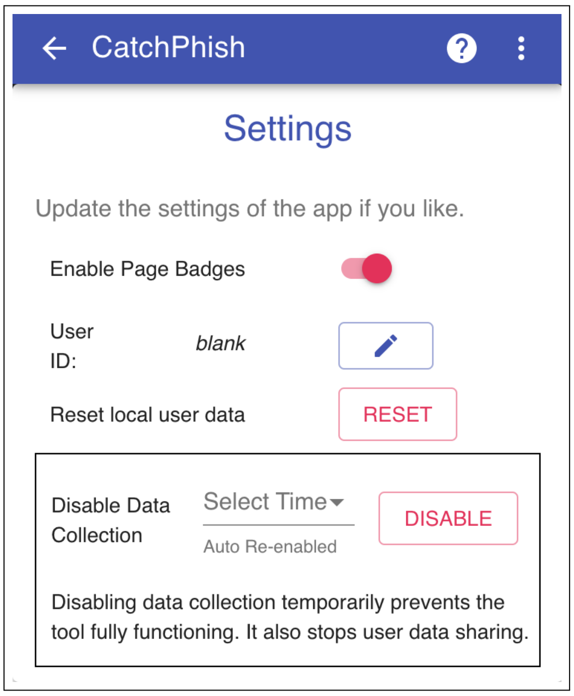
There were several additional features implemented in the browser extension, detailed in this section.

6.1.1 Gathering User Usage Data

In order to understand the effectiveness of the tool in future studies, how users interact with the tool needs to be recorded. This interaction data allows researchers to analyse the successfulness of certain aspects of the tool such as active warning click through rates, which can be compare them with existing approaches [3].[1] To facilitate this, the browser extension had to be updated to record user responses at key event points:

- Which buttons users click when presented with an active warning
- When a user adds a site to their personal white or blacklist
- Any clicks on the tool's context menu item or extension icon (events which open the extension analysis popup)
- When users click to see further details of any URL
- A change to any of the settings options set by users

[1]An active warning occurs when users try to visit a malicious site; an intervention page appears, alongside the tool's popup, to dissuade them from continuing.



Settings

Update the settings of the app if you like.

Enable Page Badges

User ID: *blank*

Reset local user data RESET

Disable Data Collection Select Time Auto Re-enabled DISABLE

Disabling data collection temporarily prevents the tool fully functioning. It also stops user data sharing.

Figure 6.1: The browser extension's settings menu.

To gather this recorded data, the Analysis Server API was updated to store information labelled as user events. A bespoke database table was created to store this information, with each record including the user's ID.

6.1.2 User Experience Settings

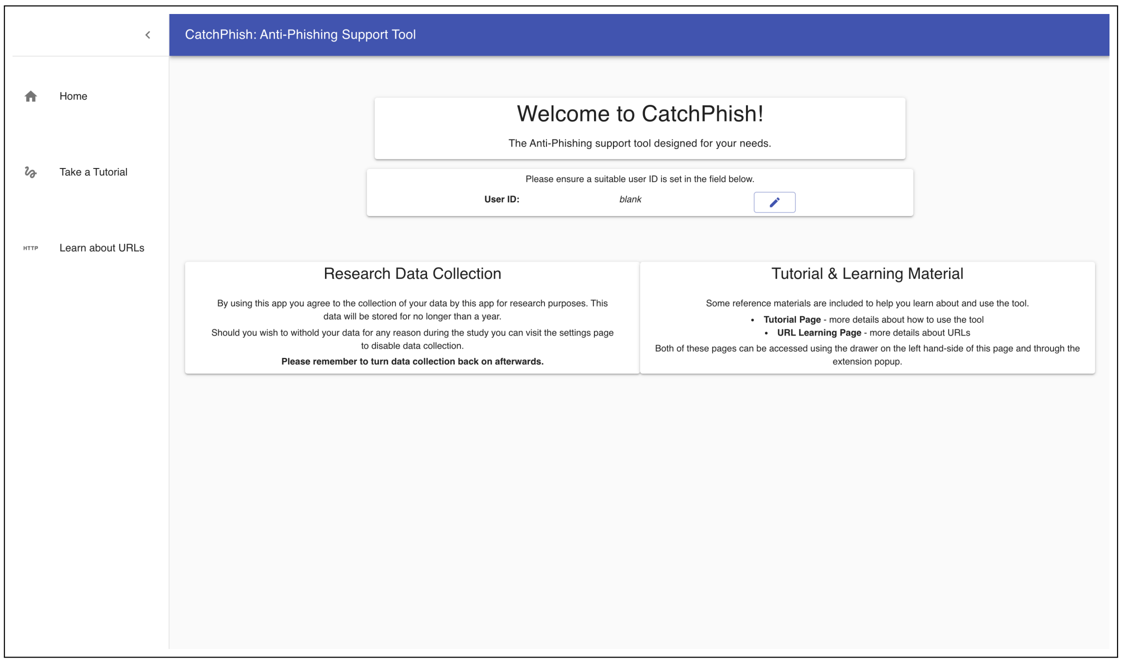
As part of some quality of life and data protection improvements for users, various features were implemented as part of the extension settings (see Figure 6.1).

The abilities were:

- to disable communication with the server for specified periods of time (a limited user data control method).
- to enable/disable the passive warnings displayed beside each link on a page.[2]
- the backend implementation required to send an email to the developers for feedback purposes.[3]
- selection of a user ID for data gathering (to easily provide users a shared ID across multiple extension instances).

[2]The passive warnings were previously implemented. These are traffic-light coloured badges corresponding to the URL classification. The disablement of these warnings does effect the occurrence of active warnings when users try to visit malicious URLs.

[3]The UI for this feature was completed the previous year.



CatchPhish: Anti-Phishing Support Tool

Welcome to CatchPhish!

The Anti-Phishing support tool designed for your needs.

Please ensure a suitable user ID is set in the field below.

User ID: *blank*

Research Data Collection

By using this app you agree to the collection of your data by this app for research purposes. This data will be stored for no longer than a year.

Should you wish to withold your data for any reason during the study you can visit the settings page to disable data collection.

Please remember to turn data collection back on afterwards.

Tutorial & Learning Material

Some reference materials are included to help you learn about and use the tool.

- **Tutorial Page** - more details about how to use the tool
- **URL Learning Page** - more details about URLs

Both of these pages can be accessed using the drawer on the left hand-side of this page and through the extension popup.

Figure 6.2: The browser extension landing page implementation.

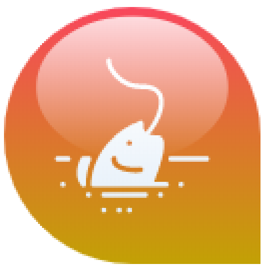


Figure 6.3: The new browser extension icon.

Future research studies utilising this platform intend to record all URLs that a user visits. Since this makes the existing implementation of a URL cache in the browser extension redundant, this cache was re-purposed to allow the extension to provide analysis on already retrieved URLs during periods where the user has disabled communication with the server.

6.1.3 Landing Page Implementation

A dedicated landing page was implemented as part of the tool to welcome users to the site. This landing page is displayed the first time users install the tool. It is intended to highlight data storage and usage information, provide users a link to the tutorial and URL information pages (developed last year) and allow users to set their user ID.

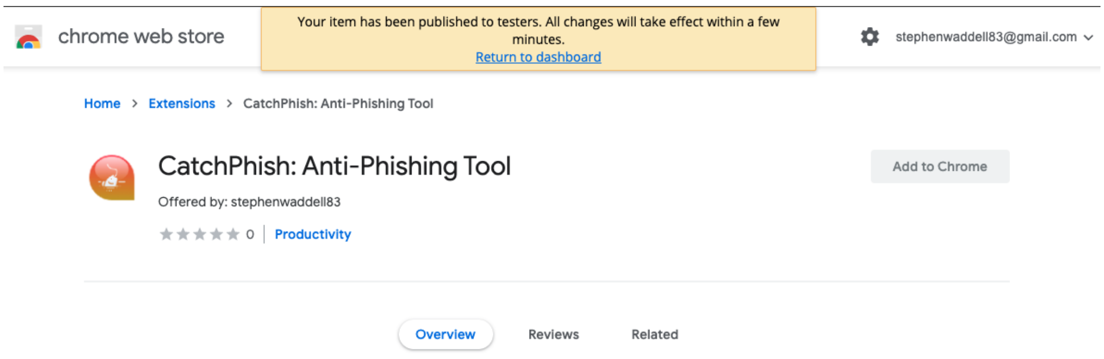

chrome web store
Your item has been published to testers. All changes will take effect within a few minutes.
Return to dashboard
stephenwaddell83@gmail.com
Home > Extensions > CatchPhish: Anti-Phishing Tool
CatchPhish: Anti-Phishing Tool
Offered by: stephenwaddell83
0
Productivity
Add to Chrome
Overview
Reviews
Related

Figure 6.4: The browser extension's page listing on the Chrome Web Store.

6.2 UI Improvements

There were several UI feature improvements highlighted by users in previous evaluations. These required a serious of small fixes and changes. An example of these is user confusion between the option to 'Report an Issue' to developers and 'Report a URL' to a blacklist. This was resolved by renaming the blacklisting option.

One requested feature from previous evaluations was an icon to replace the default Chrome Extension icon; the new icon is illustrated by Figure 6.3.

However, several features required more dedicated bug fixes. For instance, a bug was found with the intervention page - which was hard-coded to return users to a known safe site (*google.com*) if they clicked the wrong sequence of buttons. This was fixed by ensuring the users were consistently returned to the page they were previously on prior to intervention. Each of these bugs was iteratively identified and improved.

6.3 Deployment

To facilitate future studies, the tool was deployed on the Chrome Extension store. This has the ability to limit usage to select test users before full deployment, which ensures only valid participants are able to use the tool.

Figure 6.4 illustrates the store listing. This listing includes a brief description of the tool, along with an outline of data privacy considerations, to facilitate the tool's deployment.

6.4 Summary

This chapter highlights improvements made to the existing components of the project. It outlines additional features added to the system to facilitate future studies, such as user data gathering and user data control settings. It also describes the approach taken to improve the UI features and deploy the browser extension.

Chapter 7

Algorithm Fine Tuning

Several fine tuning steps had to be taken to ensure the algorithm was ready for deployment. This chapter focuses on these fine tuning steps, with a particular emphasis on threshold selection and algorithm evaluation. The chapter also discusses performance optimisations applied to the URL Processing application.

7.1 Popularity Threshold Selection

For heuristics dependent on continuous data, thresholds had to be set to output appropriate severity levels; most of which where able to be sourced from existing research [7]. However, it was not possible to find thresholds for heuristics depending on popularity indicators as these are dependent on numerous factors - including the quality of their APIs. These thresholds therefore had to be derived separately in order to be included in the system. Popularity API data is used in two sets of heuristics: safety metrics and page indicators. The popularity APIs are provided by Alexa Top Sites (for the Global Popularity and Sites Linking In heuristics), OpenPageRank (for the PageRank heuristics) and SharedCount (for the Social Media Reputation heuristics).

7.1.1 Methodology

I carried out an experiment to assess the response distribution of the popularity APIs for both safe and malicious URLs. Both sets of URLs were evaluated to correspond with the two sets of heuristics depending on popularity indicators.

10,000 URLs were selected for use in this study, with 5000 each being drawn from a popular whitelist (Tranco) and a blacklist (hpHosts). Random selection, using a fixed random seed, was used to select from the ranked Tranco list to ensure the whitelisted URLs would not be unfairly weighted towards the highly popular sites.

By testing the popularity APIs, two types of data were gathered: the responsiveness of the APIs and the range of values each would return when presented with a URL of a particular type. An API was determined to be unresponsive for a particular URL if it returned null or unexpected values for a particular field. For example, where the Alexa Top Sites' *Alexa Traffic Rank* field produced a value of 'No Data'. This responsiveness data was analysed to determine if there was a clear difference between how the APIs responded for whitelisted and blacklisted URLs. This type of API responsiveness is useful for indicating how much information an API has on a particular URL, indicating qualities such as whether it has been indexed yet. When this data is aggregated it can be used to infer common API characteristics, such as whether the APIs contain more data on whitelisted URLs.

The range of valid responses returned by the APIs were also analysed. Each API has its own set of units for each particular field which cannot be easily compared. Therefore, analysis was focused on the distribution of results across each individual API field, with the API fields filtered by those best able to facilitate the heuristics.

Both the responsiveness data and value analysis were combined to produce two sets of thresholds for use in the URL Processing application. The whitelist thresholds are used for the safety metrics, and the blacklist thresholds for the page metrics. Each of the thresholds were selected by analysing the distribution of results of the white and blacklist responses. The intention was to minimise the number of misclassifications for either type: safe or otherwise.

7.1.2 Results

One of the initial findings was the differing responsiveness of the APIs for white- and black- list data, indicating whether the API contained data on the URL, as seen in Figure 7.1. The APIs tended to have a significantly lower responsiveness for malicious URLs. Excluding the Social Media Reputation API, where most fields have no response for either type and otherwise return values of zero at a minimum.

Following the responsiveness findings, the API fields were filtered to retain the result distributions for the most significant numeric fields. For instance, the Alexa Top Sites' *Top Country* fields were removed. This set of fields conveyed similar information to the *Alexa Traffic Rank* field, which was of greater utility as a global ranking less likely to fluctuate as a result of regional variations.

The distribution of API responses can be seen in Tables 7.2a and 7.2b. The respective distributions indicate several differences between the different types of URLs. For instance, the significantly higher average traffic rank for blacklisted URLs as opposed to whitelisted URLs. Differences in distribution between these respective URL types are used to produce the thresholds. These results are combined with knowledge from prior research findings, such as that popular URLs tend to have a low traffic rank.

Tables 7.3a and 7.3b illustrate the selected thresholds, each of which is individually justified. The thresholds were only selected for the most representative API fields to simplify their corresponding heuristics. The small overlap in the distributions of the white- and black- listed URLs was considered when selecting the whitelist thresholds. The potential false positive misclassifications of phishing URLs caused by the selection

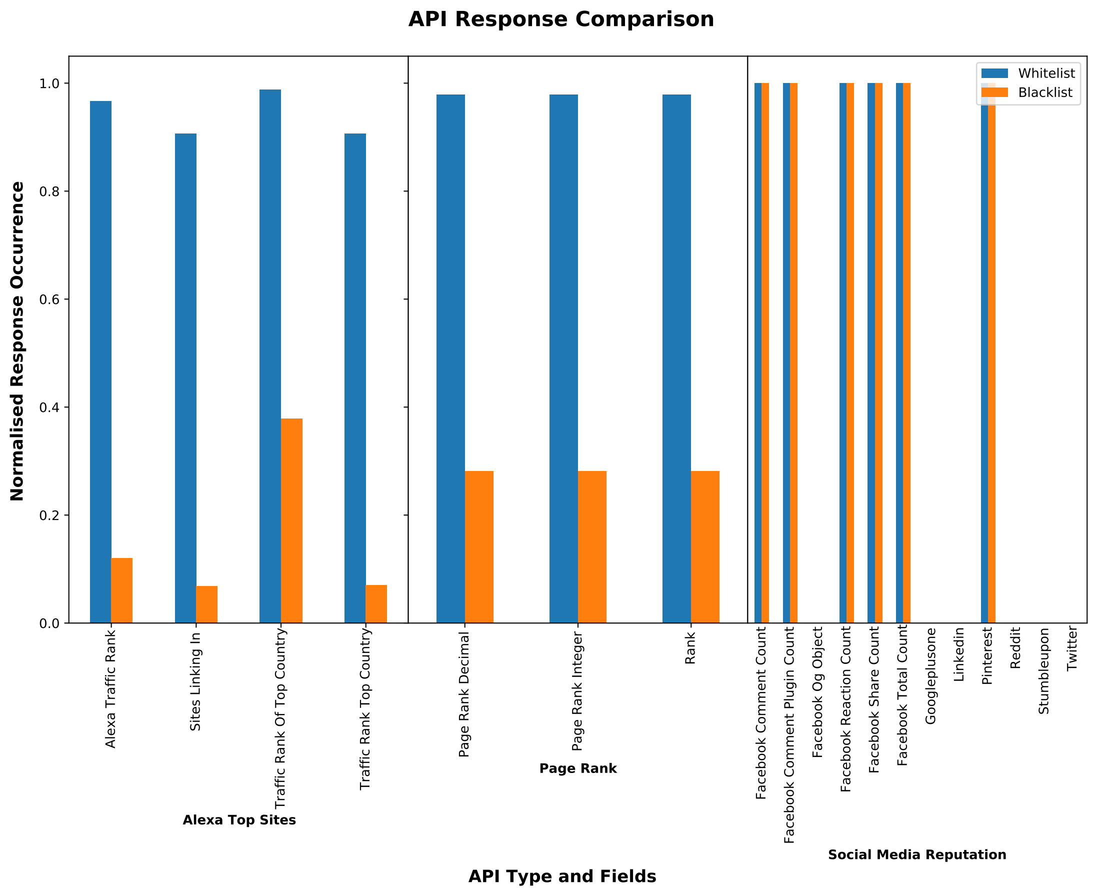


Figure 7.1: Comparison of normalised API response occurrence for the popularity APIs. Two sets of 5000 URLs from white- and black- lists were each processed by the APIs, with the results aggregated to indicate how much data each API contained for URLs of each particular list type. The API data is further broken down by how much data is available for each list type for each API field.

	API Field	Count	Mean	Std Dev	Min	25.00%	50.00%	75.00%	Max
Alexa Top Sites	Alexa Traffic Rank	4833	208173.1	912351.9	4	6178	18202	64235	10913986.0
	Sites Linking In	4531	4542.0	56595.1	1	342.8	1235.5	2958.3	3457310.0
	Traffic Rank of Top Country	4939	11590.4	25106.7	1	322.5	2447	10886.8	271247.0
Page Rank	Page Rank Decimal	4895	4.5	1.2	0	4.2	4.7	5.1	10.0
Social Media Reputation	Facebook Comment Count	5000	57027.1	2524913.5	0	25	406	2174	161536782.0
	Facebook Comment Plugin Count	5000	5.9	280.9	0	0	0	0	19373.0
	Facebook Reaction Count	5000	361521.7	15657320.3	0	53.5	954	5887	958591658.0
	Facebook Share Count	5000	492394.5	20907001.4	0	209.5	2730	10989	1294585871.0
	Facebook Total Count	5000	910901.8	37249049.3	0	367	4673	21654.5	1881373120.0
	Pinterest	5000	665.7	28761.0	0	0	0	0	1747521.0

(a) Whitelist Responses

	API Field	Count	Mean	Std Dev	Min	25.00%	50.00%	75.00%	Max
Alexa Top Sites	Alexa Traffic Rank	601	2567582.0	3272606.7	996	3519	557177	5099097	10266800.0
	Sites Linking In	340	11479.0	10574.2	2	719.0	15624.0	15624.0	43953.0
	Traffic Rank of Top Country	1894	2251.5	12197.9	1	2.0	4	32.5	123820.0
Page Rank	Page Rank Decimal	1404	1.6	1.1	0	0.7	1.7	2.2	4.3
Social Media Reputation	Facebook Comment Count	5000	43.2	492.0	0	0	0	0	8827.0
	Facebook Comment Plugin Count	5000	0.0	0.3	0	0	0	0	9.0
	Facebook Reaction Count	5000	273.3	7066.8	0	0	0	0	216257.0
	Facebook Share Count	5000	183.7	1500.7	0	0	0	5	30491.0
	Facebook Total Count	5000	500.2	8517.7	0	0	0	11.25	255575.0
	Pinterest	5000	0.0	0.0	0	0	0	0	0.0

(b) Blacklist Responses

Figure 7.2: The distribution of results across popularity API Fields for both white- and black- lists. Each of these distinct API field has it's own set of units so therefore cannot be easily compared. For example, the Page Rank API's *Page Rank Decimal* field has a decimal value which represents a URLs page rank normalised into a decimal range between an interval of 0 and 10.

	API Field	Heuristic (Safety Metric)	Threshold	Justification
Alexa Top Sites	Alexa Traffic Rank	High Global Popularity	64235=<	Based on the whitelist 75th percentile, therefore includes the top 75% of tested whitelisted sites.
	Sites Linking In	High Sites Linking In	>=4542	Despite there being a small minority of blacklisted sites with high sites linking in, threshold set to be greater than or equal to the whitelist mean.
Page Rank	Page Rank Decimal	High Page Rank	>=4.5	Includes any sites greater than or equal to the whitelist mean.
Social Media Reputation	Facebook Total Count	High Social Reputation	>=4673	Based on the whitelist 50th percentile, therefore includes the top 50% of tested whitelisted sites.
	Pinterest		>=665	Includes any sites greater than or equal to the whitelist mean.

(a) Safety Metric Thresholds

	API Field	Heuristic (Page)	Possible Threshold	Known Threshold	Justification
Alexa Top Sites	Alexa Traffic Rank	Low Global Popularity	>=557177	No Result	Possible threshold is based on the blacklist 50th percentile, therefore includes the top 50% of tested blacklisted sites. The Known threshold is based on the consistency of the API for whitelisted URLs.
	Sites Linking In	Low Sites Linking In	N/A	No Result	No Possible threshold set due to the high mean compared to the whitelist responses. The Known threshold is based on the consistency of the API for whitelisted URLs.
Page Rank	Page Rank Decimal	Low Page Rank	2.2 < x =< 4.3	No Result or (2.2=<)	Possible threshold sis et to include responses less than the max. The Known threshold is based on a combination: No Result due to the consistency of the API for whitelisted URLs; less than 2.2 to include the top 75% of blacklisted URLs.
Social Media Reputation	Facebook Total Count	Low Social Reputation	11.5=<	N/A	Possible threshold based on the blacklist 75th percentile, therefore includes the top 75% of tested blacklisted sites. Known threshold is not set due to unreliablity of the API - the consistency of zero value responses.
	Pinterest		N/A	N/A	Neither threshold is set due to the unreliablity of the API for the Pinterest field - the consistency of zero value responses for both white and black lists.

(b) Page Heuristic Thresholds

Figure 7.3: The thresholds selected for the popularity API Fields for whitelists and black- lists. These API fields have been filtered by the values most useful for facilitating their corresponding heuristics. The selection of thresholds for each field is individually justified with comparison to both the API responsiveness Figure 7.1 and the valid response distributions Figure 7.3. The blacklist results are used to produce page heuristic thresholds for both 'possible' and 'known' severity levels where relevant.

of the whitelist thresholds was intended to be compensated by the differing weighting of the popularity heuristics in the safety algorithm.

7.2 Expert Heuristic Evaluation

To ensure the heuristics had been implemented with an appropriate level of accuracy, an expert in phishing heuristics was consulted. The chosen expert was a member of the TULIPS Group who has published several papers on phishing and URLs.

Expert evaluations involve using the experience of experts in a field to analyse potential problems in proposed designs. As an evaluation method it is useful for evaluating difficult material. This method can help identify any user interface or technical issues in design processes before employing more costly evaluation methods [64].

The basis of this expert evaluation was to assess the implemented heuristics, focusing on three key components: their severity levels, their thresholds and their user-facing explanations.

7.2.1 Methodology

An interview was arranged with the expert in order to analyse the heuristic implementation. During this interview, the expert was presented with the source code for the heuristics: including the appropriate threshold values and associated severity levels. They were also sent the list of user-facing sentences.

During the interview, detailed notes were taken to record the expert's feedback. These were analysed to produce qualitative findings covering both positive and negative aspects of the heuristic selection. The expert's recommendations were then used to improve the quality of the implemented heuristics with the intention of improving both the system and algorithm performance.

7.2.2 Results

The expert thought the choice and implementation of heuristics was of a particularly high quality, with the associated user statements appropriately written. They were also confident in the underlying logic behind the the safety and classification algorithms. However, they suggested modification of several heuristics which were incorporated into the system to better suit the latest phishing trends.

Firstly, they questioned the use of the 'hyphen' character as part of an implemented atypical deliminator heuristic. They pointed out that attitudes towards hyphens in URLs had changed, becoming much more prevalent in legitimate URLs, and it would be better to merge the atypical deliminator heuristic with the suspicious character heuristic. They also suggested implementing a distinct heuristic for the number of hyphens in a URL - which is a known phishing feature.

They also presented a similar argument for a previously implemented heuristic evaluating the use of suggestive word tokens in URLs. They highlighted how security keywords such as ‘confirm’ used to be malicious indicators, but that this has changed due to their increasing legitimate usage. The suggestive word token heuristic was therefore excluded from the system.

Secondly, they suggested several changes in severity for particular heuristics. The search result heuristic in the page heuristics was an example they recommended be updated. They suggested this heuristic would be a more representative feature if it were to evaluate the search results for both the hostname and domain, rather than just the domain. They recommended that the presence of the hostname and domain should be used as possible and known thresholds respectively (since a lack of domain presence suggests no prior search indexing, whereas a lack of hostname presence suggests a lack of popularity). They also recommended a change to the safety search result heuristic to focus on search results for the hostname rather than the domain, since this is a more representative feature of safe URLs.

These are prime examples of the detailed and helpful feedback the expert provided. Overall, they were impressed by the system and the quality of work involved. The feedback from this expert was utilised to produce the the final set of heuristics and associated severity levels illustrated in Figure 5.4.

7.3 Iteration with Real Data

To evaluate the algorithm and system in a deployed setting, a trial with real data was planned. The intention of this trial was to use particularly challenging datasets to evaluate the performance of the algorithm. This intention of this trial was to evaluate the successful implementation of the heuristics and provide an initial indication of the general algorithm performance.

Challenging URL data sets were selected for this trial as part of a functional testing approach to model the robustness of the overall system. Whilst the chosen datasets were atypical of realistic system use cases, they contained more unusual test cases which were useful for testing the functionality of individual heuristics. This approach was intended to allow the heuristics to be better tuned by ensuring each of the heuristics were functioning as expected on a wide range of inputs. Thus evaluating the system with these datasets was intended to result in better system robustness and algorithmic performance.

7.3.1 Methodology

This study uses the system to analyse previously classified URLs in order to estimate the system’s effectiveness. 10,000 URLs were used for evaluation, with 5000 being drawn from a whitelist - ParaCrawl and a blacklist - Phishbank, neither of which are used in the system. Both of these lists were composed of URLs, rather than lists of domains, allowing me to test the full range of the tool's functionality.

To prepare for this study, a test suite was written to process each URL and send it to the server for analysis. The classification results were then matched to the ground truth to evaluate the overall accuracy of the system. The results were then examined in further detail to understand the effects and performance of the individual heuristics. The performance of these heuristics was analysed with an understanding of their expected performance as outlined in literature [7]. There was a focus on how likely particular heuristics are to activate for each test set.[1] The URL classifications were then grouped by their classification reasons to better understand how the algorithm utilises these heuristics at a higher-level.

The results of this trial were intended to be subsequently utilised to improve the implementation of the heuristics and the performance of the algorithm in general.

7.3.1.1 Dataset Details

The ParaCrawl [22] dataset was created for the Connecting Europe Facility as part of a project to harvest parallel data from the Internet to help translate between languages used in the European Union. This resource includes multiple webpages which have different translations across the 23 official languages of the European Union. As the characteristic of having multiple translations is a strong indicator of a safe site, the list of URLs sourced from this corpus can functionally be used as a whitelist.

However, ParaCrawl is a challenging whitelist for several reasons. Firstly, this dataset was not originally developed as a whitelist. As a result, the URL whitelist produced from this corpus does not contain the highly popular global URLs which users typically visit. As the system's safety classification algorithm relies on popularity metrics for identifying safe sites, this dataset is a significant challenge. Many of the sites included in this dataset are not particularly popular worldwide since the corpus is focused on regional European rather than global sites. Another challenge presented by this dataset is the differing syntax of URLs depending on the origin country of the URL. Whilst the hostname of the URLs largely remain consistent with ASCII and English only characters, there is a higher prevalence of non-ASCII characters in non-English URLs. This dataset therefore allowed us to model the importance of the popularity heuristics for classifying safe sites (the algorithm's false negative cases), along with the effectiveness of the lexical heuristics.

The Phishbank dataset was sourced from a list of archived phishing URLs stored on the Phishbank platform [47]. This is a challenging dataset as many of the blacklists incorporated into the system such as hpHosts and PhishTank filter out URLs which are not currently active. This means Phishbank URLs are less likely to be identified by the blacklist presence heuristics used by the system. The blacklisting heuristics were therefore expected to be less significant contributors to the URLs' final classifications.

[1]In this discussion, a heuristic is deemed to have activated if it outputs any of it's associated severity levels.

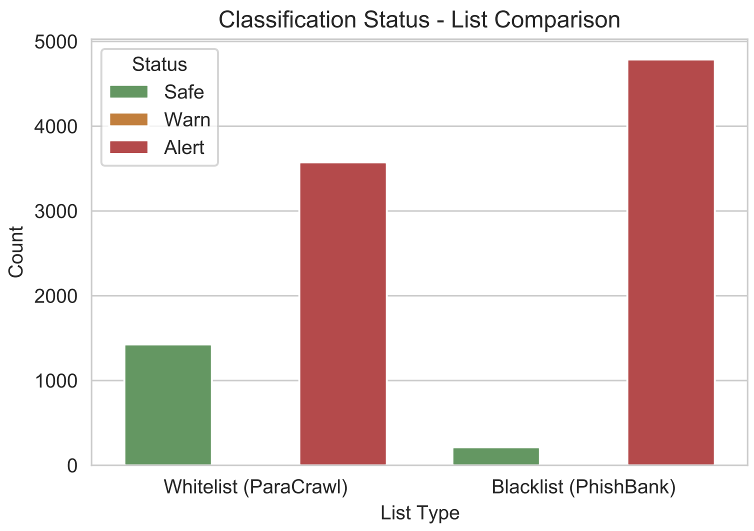


Figure 7.4: The classification status is broken down for both the ParaCrawl whitelist and the PhishBank blacklist, displaying the final classification status of each URL.

This allowed us to test the effect of the more diverse set of heuristics for classification beyond the blacklist presence heuristics.

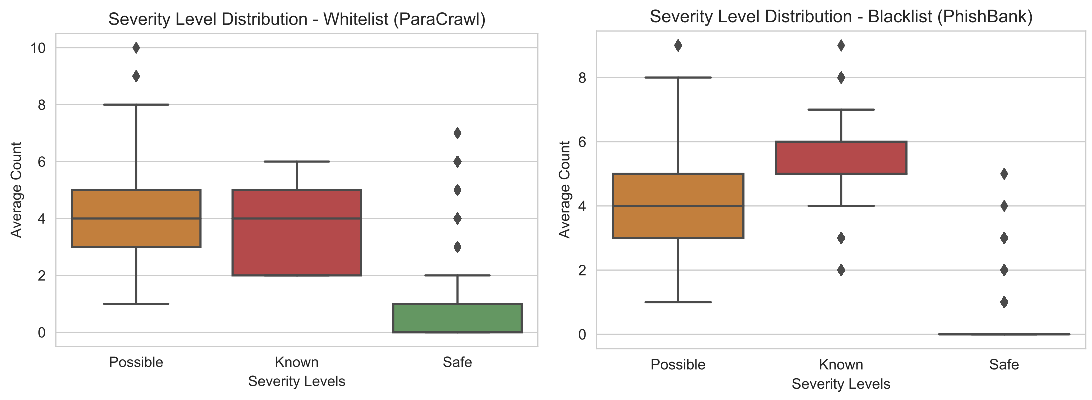
7.3.2 Results

The system was found to have a mixed performance on the distinct list categories. Whilst the system performed particularly well on blacklisted URLs, it was found to have high false negative rate when classifying URLs sourced from the whitelist. The performance of individual heuristics was found to be partially responsible for the high false positive rate. However, the misclassification of URLs from the ParaCrawl dataset was found to be a greater consequence of the safety algorithms focus on popularity metrics. These results are outlined in detail in this section, with several further supporting graphs in Appendix D.

7.3.2.1 Overall Algorithm Accuracy

Whilst the the system was able to accurately classify malicious phishing URLs with regularity, there were a high number of false negatives; with the whitelisted URLs being classified as malicious more than nought, as illustrated by Figure 7.4.

When analysing the results by severity level, as in Figure 7.5, there was a higher proportion of heuristics activating with a ‘known’ severity level for blacklisted rather than whitelisted URLs. Conversely, a higher proportion of heuristics activated with a ‘safe’ severity level for whitelists. Both of these findings were expected outcomes, however, the amount of heuristics which produced ‘known’ activations for whitelisted URLs was abnormal. Whilst the high false negative rate was not a particularly unexpected



(a) Whitelist (ParaCrawl)

(b) Blacklist (PhishBank)

Figure 7.5: The severity level distributions for the individual heuristics for both the whitelists and blacklists.

outcome, the high amount of 'known' activations suggested issues with the implementation of the malicious indicators.

There was also an absence of URLs classified with a final 'warn' classification results. It was an unexpected outcome to see almost no URLs with this classification for either this, whilst a limited amount of warn' classifications might have been an expected outcome given the limited amount of heuristics outputting a 'possible' severity level. This might be partially explained by the composition of the algorithm however, with it only requiring the occurrence of one 'known' activation for a URL to be marked as malicious.

7.3.2.2 Performance of Individual Heuristics

By analysing the activation of individual metrics as presented in Figure 7.6 and Figure 7.7, several patterns were identified. Foremost of these was the activation of the Different Top Level Domain and Typosquatting heuristics for almost every URL for both white- and black- lists. This suggested significant errors in the implementation of both of these heuristics, given their low expected occurrence outlined in prior research [7].

As a result of implementation errors of certain heuristics, there was a limited amount of inference which could be made about the underlying datasets given the unreliable performance of the algorithm. For example, it was particularly challenging to understand the effect of heuristics such as Registrant Name Hidden on the datasets. Whilst the high amount of activations of this heuristic that occurred for both list types may typically indicate it is an insignificant phishing feature, this is a challenging conclusion to have had confidence in given the errors in other heuristics. The amount of suspicious characters in the URL followed a similar pattern. This might have typically been explained by the multi-language composition of the ParaCrawl dataset and the generally high amount of suspicious characters in phishing URLs. However, the high activations might also have been a consequence of implementation errors. These results therefore highlight the utility of testing the system in this functional manner with these challenging datasets: by evaluating how well each heuristic meets its intended requirements.

However, general patterns were still able to be identified from groups of heuristic activations. The lack of significant proportions of ‘safe’ activations for whitelisted URLs is one example. The number of activations of the Presence in a Whitelist heuristic was found to be too limited to compensate for the under-performance of the popularity safety metrics. This also highlighted a limited presence of ParaCrawl URLs in the system’s whitelists. The impact of the popularity heuristics on the overall classification performance was also found to be minimal as originally expected, with only one popularity heuristic typically activating for each URL.

7.3.2.3 Underlying Heuristic Effectiveness

To model the effect of suspected incorrect classifications, the Different Top Level Domain and TypoSquatting results were removed from the analysis results and the URLs were reclassified. This produced a similar result as in Figure 7.4. Breaking the results down by classification reason, Figure 7.8 illustrates the significant number of classifications which still occur due to ‘known’ classifications.

These results highlighted the limited impact of the two broken heuristics on the overall performance. Therefore, the conclusion was drawn that the whitelisted URLs are being misclassified due to the general performance of the algorithm on this dataset.

Whilst a more minor issue, Figure 7.4 also highlighted a small number of false positives when identifying phishing URLs. As highlighted in Figure 7.8 this was a result of the phishing URLs being marked as safe by safety algorithm rather than a failure of the malicious indicators.

Whilst there was a small minority of blacklisted sites with extended validation certificates, the primary finding was the amount of URLs present in whitelists as highlighted in Figure 7.6. All of the URLs discussed had domains present in the Tranco list, which suggests significant issues with the formation of this list. Further the reliability of whitelists more broadly might be questioned since Tranco itself is a cumulative whitelist. Therefore, this work raises potential future research avenues through the questions it raises on the reliablity of whitelists.

7.3.3 Impact on System Implementation

As a result of the unusual performance of several heuristics such as the Typosquatting and Different TLD heuristics, the implementaton of each heuristic was iteratively evaluated to identify any potential bugs. This process also resulted in minor changes to the approach of several heuristics, such as the filtering of country level TLDs when analysing whether a URL has a Different TLD from highly popular URLs. By identifying bugs and optimising the performance of individual heursitics, the intention was

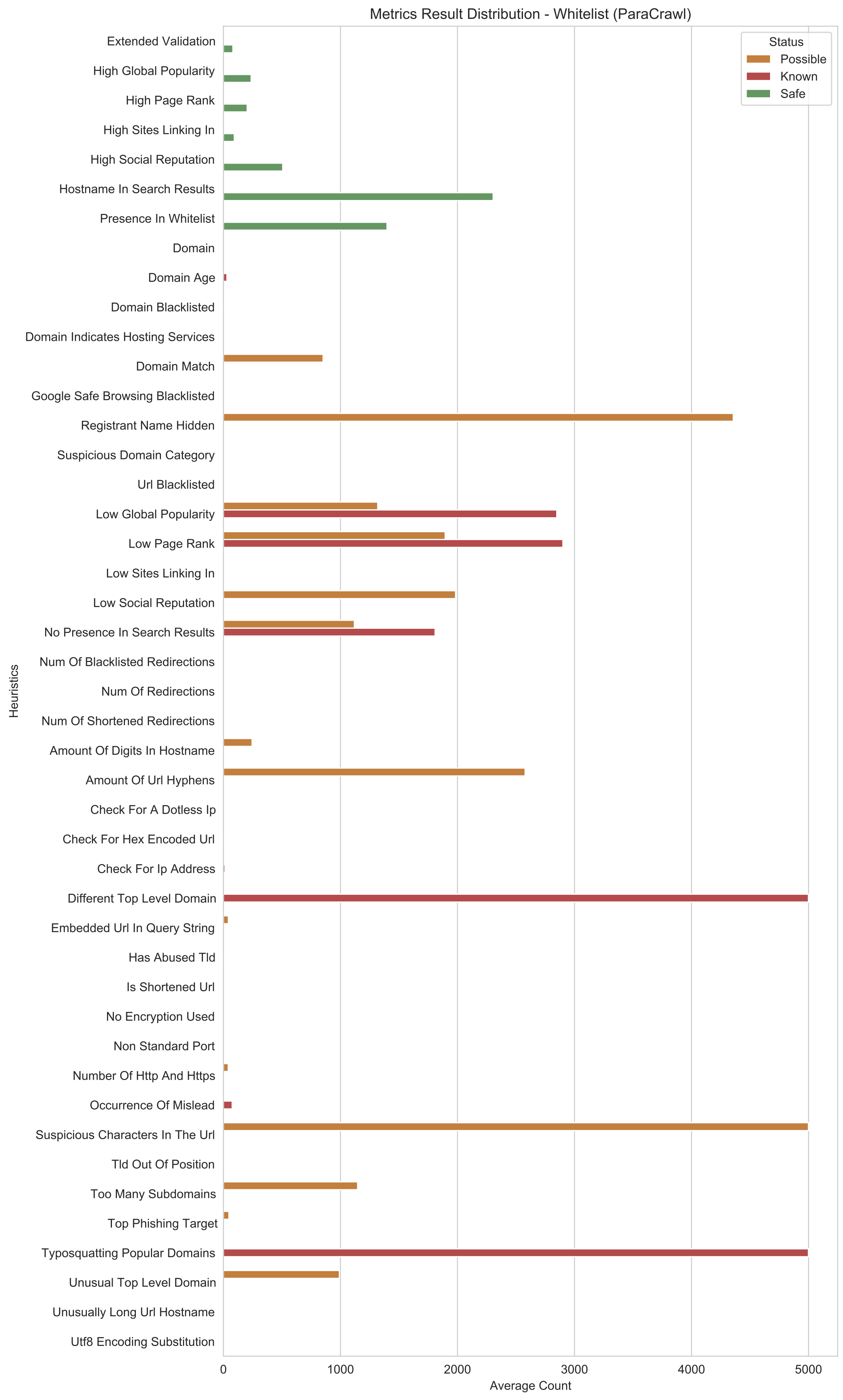


Figure 7.6: The whitelist results are broken down by individual heuristic. Each heuristic is displayed with the number of times it activated with it’s associated severity levels.

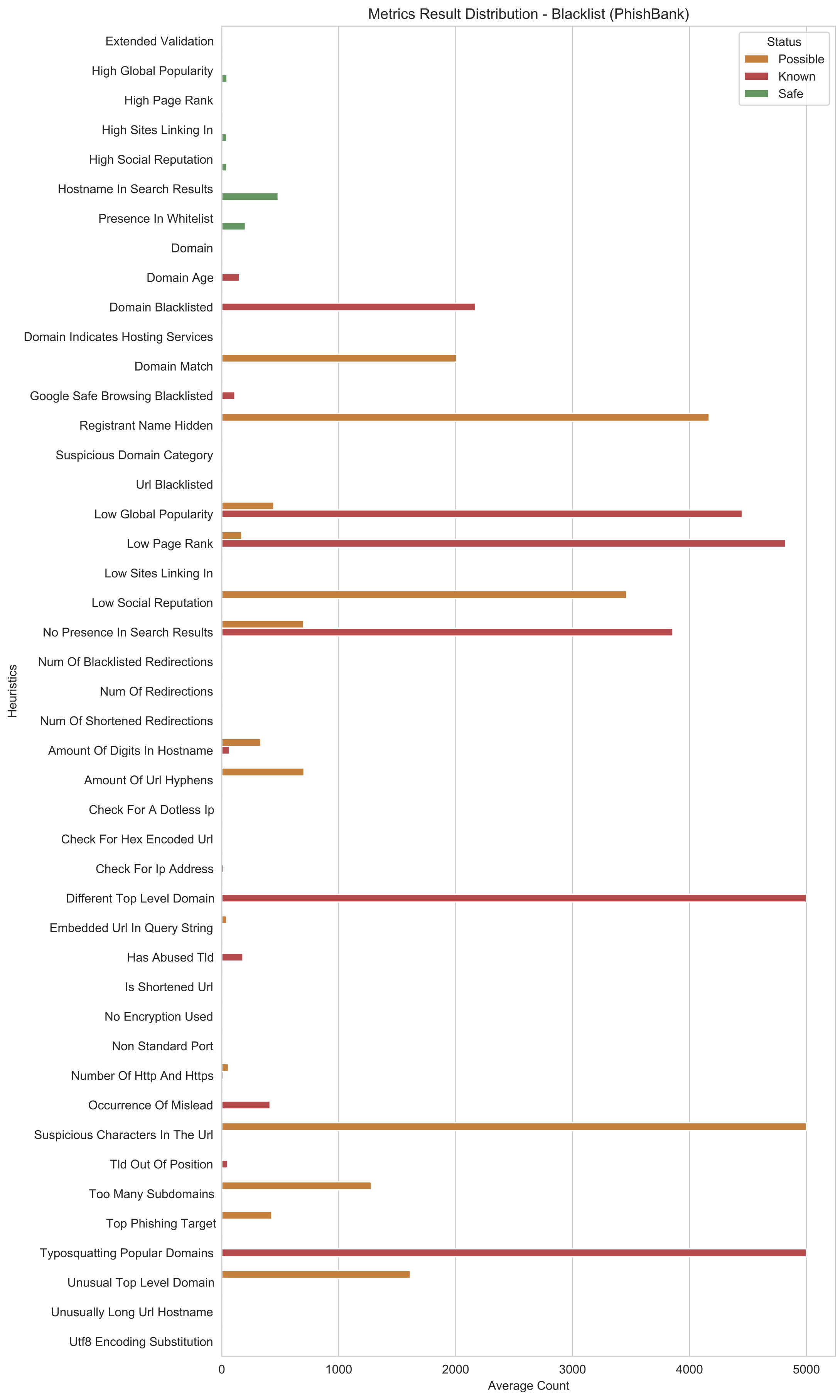


Figure 7.7: The blacklist results are broken down by individual heuristic. Each heuristic is displayed with the number of times it activated with it's associated severity levels.

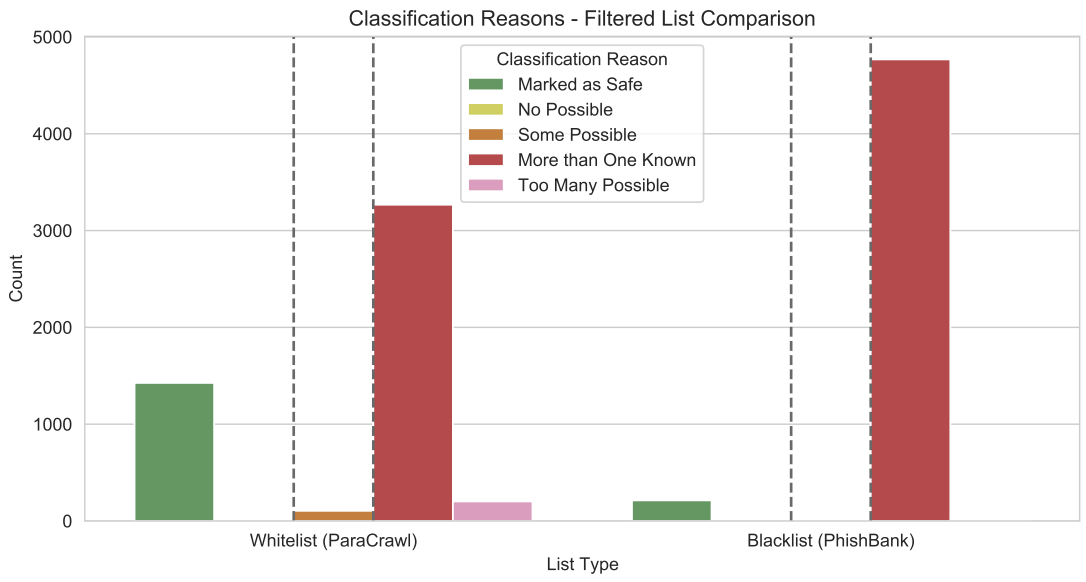


Figure 7.8: The URLs are grouped by how they have been classified, represented by their colour, the URLs are further grouped by the reason for their classification.

to produce a system which would be able to draw more reliable conclusions from analysis.

One of the key considerations was improving the false negative rate of the system following it's performance on the ParaCrawl dataset. Since it was a significant challenge to alter the impact of the malicious heuristics without undermining the true positive rate, the emphasis was on solutions which improved the safety classification of the URLs.

Therefore, to improve the amount of safe URL classifications, two heuristics were added to the tool's safety metrics: Presence in Categorised Data and Fortiguard Safe Domain. To implement the former heursistic, the DMOZ categories site list and UK web archive were parsed and added to the system database. Domain categorisation data gathered from Fortiguard [35] for the classification of the domain heuristics was utilsed as part of the Fortiguard Safe Domain metric. This was possible as the Fortiguard service further categories it's domain categories into safe and malicious categories.

Categorised data lists are effectively whitelists, being vetted by multiple users and publically available the Presence in Categorised Data was utlised as a primary safety metric. However, as the Fortiguard Web Filter service was found to be produce unreliable results for malicious URLs after initial testing on blacklisted URLs, this was only utilised as a secondary supporting category. The Fortiguard Web Filtering service would mark blacklisted URLs from the Phishbank dataset with a non-malicious category approximately 50% of the time. This suggests several deficiencies in the Fortiguard service, which might an interesting focus of future work.

The addition of these heuristics and associated user statements was evaluated by the same expert from Section 7.2. They found both the heuristics and user statements to be appropriate additions to the system to account for the results.

There was no further changes to the implementation made to handle the small amount of identified false positives resulting from blacklisted URLs found to be present in whitelists. This was a conscious decision made due to the limited false positive rate, with an understanding of the potential negative effect on the false negative rate if the use of whitelists were to be changed in any way.

7.4 Performance Optimisations

One of the performance optimisations that was a consideration when designing the system was the addition of asynchronous IO. This was thought to be a way of improving system efficiency by better handling the multiple data source accesses required to process each URL. This involves moving all of the system's IO operations to a secondary thread where they can be handled asynchronous and processed as each operation completes. This improves application performance by freeing up the CPU to process alternate tasks instead of waiting on blocking IO operations. The major IO bottlenecks in this system were a result of the IO waiting times caused by the number of database accesses and web requests required to process each URL.

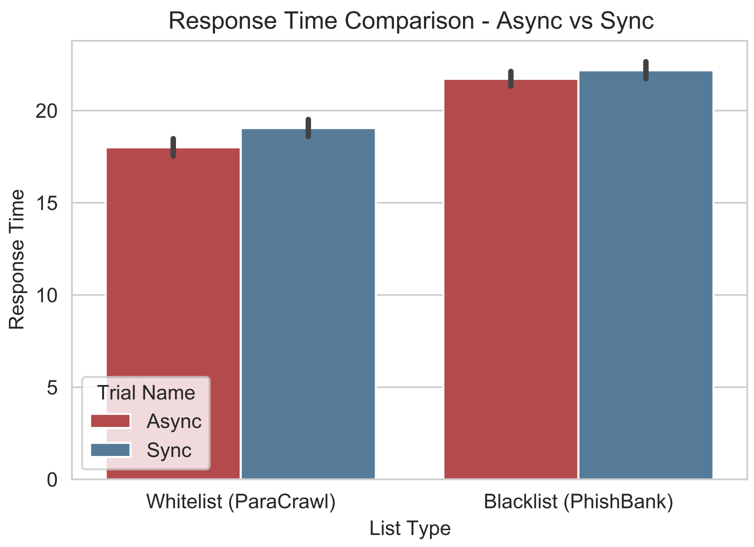


Figure 7.9: The URL Processing app response times: synchronous and asynchronous implementations. These results were gathered client-side and therefore include the network latency of communicating with the server.

A response time trial was carried out to evaluate the implementation of this feature, as async IO is known to be significantly challenging to implement correctly. The system with a finalised set of heuristics following changes resulting from the Iteration with Real Data trial was the basis of the async IO adaptions. To measure the effectiveness of the asyncIO addition the URL Processing app was trailed with both synchronous and asynchronous implementations. The asyncio, aiohttp and aiopg libraries were used to implement the async IO feature - replacing the app's control logic, Postgres database requests and web requests.

To evaluate the response time of the Analysis Server, 1000 URLs randomly selected from a whitelist and blacklist (ParaCrawl and Phishbank). This was to ensure the server would preform with a consistent time for each. The URL Processing app was deployed on the App Engine platform with each URL directly sent to the HTTP end point of this app - to reduce impact effect of the wider system on the response time.

As Figure 7.9 illustrates the async IO implementation of the URL Processing app was more efficient, so this implementation was therefore utilised in the final system.

7.5 Summary

This chapter highlights the significant efforts made to fine tune both the algorithm and system performance. An analysis into popularity heuristic thresholds is initially conducted to set the thresholds for the system. The overall heuristics, user statements and severity level thresholds are then evaluated by an expert in phishing. This evaluation highlighted the quality of the work completed and suggested several improvements which were subsequently made to fine tune the severity levels.

Following the setting of severity levels, a study was conducted to evaluate the system with real data. Challenging dataset were chosen to push the system to its limits and highlight potential bugs. This study illustrated positive performance when identifying phishing URLs but a false negative rate when analysing whitelisted URLs. The results of this study were analysed and several issues with heuristics where identified which were subsequently corrected, with the implementation of further safety heuristics added to improve safe URL classification.

The chapter lastly highlights system level performance improvements to enhancing the response time by making the system IO calls asynchronous. The addition of this feature was highlighted to be a positive addition for the overall system response time.

Chapter 8

Evaluation

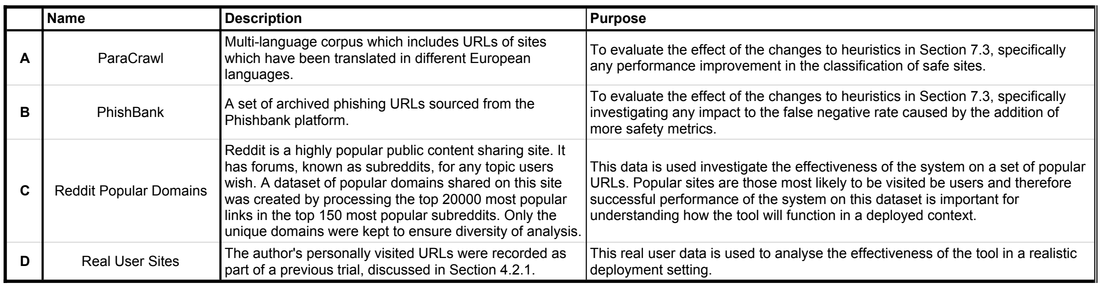
This chapter outlines the approaches used to evaluate the accuracy of the overall system and the changes made to the UI. Preparations for a future longitudinal study are also discussed, alongside this study's expected research outcomes.

8.1 URL Analysis Study

The overall goal of this study was to analyse the effectiveness of the heuristic algorithm and the wider system for the general system use cases. As part of this study, the system is used to classify a selection of datasets: composed of safe URLs, phishing URLs and URLs drawn from typical users. This is intended to evaluate the system from several use cases and estimate it's effectiveness in a deployed setting. This was the primary study used to evaluate the effectiveness of the system, intended to evaluate it according to two metrics: algorithm accuracy and system response time. The outcome of this study results in a thorough analysis of the system's utility as an analysis server, with a focus on its ability to complement the existing browser extension as part of a research platform for facilitating user anti-phishing training. This discussion, alongside the results of this study, are the focus of Chapter 9.

8.1.1 Methodology

The study uses several distinct datasets as outlined in Table 8.1 to analyse the system. All of the datasets were composed of URLs, rather than lists of domains, allowing me to test the full range of the tool's functionality. The classification results were analysed following a similar methodology as Section 7.3. A similar approach for evaluating the classification results was possible since Datasets A and B are both labelled (being a whitelist and blacklist respectively). The popularity of the links that make up Dataset C also allowed it be evaluated as a whitelist. This was also the case for Dataset D, which required analysis of each URL to determine its ground truth value, resulting in the conclusion that it too could be evaluated as a whitelist. The ground truth values



	Name	Description	Purpose
A	ParaCrawl	Multi-language corpus which includes URLs of sites which have been translated in different European languages.	To evaluate the effect of the changes to heuristics in Section 7.3, specifically any performance improvement in the classification of safe sites.
B	PhishBank	A set of archived phishing URLs sourced from the Phishbank platform.	To evaluate the effect of the changes to heuristics in Section 7.3, specifically investigating any impact to the false negative rate caused by the addition of more safety metrics.
C	Reddit Popular Domains	Reddit is a highly popular public content sharing site. It has forums, known as subreddits, for any topic users wish. A dataset of popular domains shared on this site was created by processing the top 20000 most popular links in the top 150 most popular subreddits. Only the unique domains were kept to ensure diversity of analysis.	This data is used investigate the effectiveness of the system on a set of popular URLs. Popular sites are those most likely to be visited be users and therefore successful performance of the system on this dataset is important for understanding how the tool will function in a deployed context.
D	Real User Sites	The author's personally visited URLs were recorded as part of a previous trial, discussed in Section 4.2.1.	This real user data is used to analyse the effectiveness of the tool in a realistic deployment setting.

Figure 8.1: The datasets used in the evaluation. Each of which was composed of 5000 URLs which were selected from their respective datasets using a fixed random seed.

for each URL are drawn from the knowledge of their wider datasets; this allowed individual URLs to be labelled as safe or malicious depending on whether they are from a white- or black- list.

Each of the datasets are chosen to evalute the tool in different contexts. Datasets A and B are utilised to evaluate the tool's performance in more artificial contexts in challenging environments. The challenging nature of these datasets is discussed extensively in Section 7.3.1.1. In contrast, Datasets C and D are used to evaluate the system in contexts which are more reflective of a realistic user experience - both including popular URLs with one using real user browsing data.

To prepare for this study, a test suite was written to process each URL and send it to the server for analysis. The classification results were then matched to the their ground truth values to evaluate the accuracy of the system. This is broken down to analyse the general system performance and that of individual metrics as in Section 7.3.

To evaluate the more realistic use of the system, including the browser extension and the UI Calculation components. A secondary trial was run with Selenium[1] simulating a user accessing the same URL datasets as above. This involved installing the extension on a clean Chrome Browser instance setup for use by Selenium. The instance then visited each URL using the browser; this setup allowed the extension to analyse the amount of URLs on each site. The analysis of the URLs in these pages was disabled since this was not the focus of the project. To evaluate the performance of UI Calculation statements easily, the extension logged each analysis web request. These UI statements were then matched against their corresponding heuristics and severity levels to evaluate the occurrence of any bugs. This portion of the study was also conducted with a headless browser on a virtual machine to reduce potential adverse effects from visiting the sites of malicious URLs.

Datasets C and D were used to evaluate the system response times. These datasets were chosen in particular due to their representativeness of the use case of the system. The response time results were gathered client-side and therefore inclusive of network delay. To compensate for the effect of network load at peek times, the server requests were spread across a 24 hour duration with the results across this period averaged.[2]

[1]Selenium is a portable framework for testing web applications as a playback tool [31].

[2]The network utilised by the client in all response time results had a 11.34Mbps download speed,

8.2 Expert UI Evaluation

To evaluate the added UI features and changes made as part of this work, an expert evaluation was conducted with an expert in security and secure interface design. The chosen expert was an alumni of the TULIPS Group with industrial experience as both a Security Engineer and UI Designer at large multinationals.

This expert evaluation was intended to be a smaller scale evaluation to assess the limited amount of UI changes from the prior year. Since the extension was previously evaluated using a triangulated approach and found to be a usable tool, a more extensive set of evaluations was not a necessary requirement. However, since there were some changes to the UI, an evaluation with an expert who had familiarity with last year's tool design seemed an appropriate choice.

8.2.1 Methodology

To gather results, the study was designed around presenting the final UI to the expert to capture their opinion on each aspect of the tool. They were encouraged to interact with the extension and provide feedback on the usability of the tool as a research platform. Afterwards, an unstructured interview was held, to focus on the expert's impressions of the novel additions to the UI and changes implemented in Chapter 6.

Detailed notes were taken to record the results of the evaluation. These were later analysed to produce a set of qualitative findings covering both positive and negative aspects of the UI implementation.

8.2.2 Results

The expert was, on the whole, highly impressed by the tool, expressing that the general usability of the tool was of an extremely high standard. Both the utility of the tool as a research platform and the richness of features provided for users, were aspects they felt demonstrated great potential. In their general impression of the tool's usability, they felt the extension was extremely easy to navigate, with each of the features being very clear and easy to use.

Two particular additions to the browser extension's settings impressed the expert: the ability to disable the tool's passive warnings, and the ability to disable data collection for specific periods of time. They believed these features showed considerate appreciation of the target users, which they felt would encourage uptake of the tool. Specifically, the ability to disable data collection was a feature they highlighted for commendation; particularly as it is not commonly available in other privacy invasive systems.

Whilst they felt, strongly, that the extension had fully met its usability objectives as a research platform, outwith the scope of this project, the expert also highlighted the

0.74Mbps upload speed and a ping of 33ms.

potential of the system as an application suitable for a marketable industrial level. This was based on their experience of industry security applications, suggesting that the tool would be highly useful for users beyond its applications in research. To facilitate this, they felt there could be some aesthetic improvements to some of the tool's pages to bring them to more of a marketable level.

Specific improvements they highlighted were to enhance the aesthetics of the Landing Page displayed in Figure 6.2. They felt this could be improved with the addition of the tool's logo and further graphics to make it more visually attractive to users. They also felt the navigation menu on this page, whilst meeting all functional requirements, could be further polished by adding subheadings for each of the topics in the tutorial pages.

Overall, they were extremely impressed by the extension and expressed an enthusiasm to use the tool themselves should it be more widely deployed in the future.

8.3 Planning for Longitudinal User Study

This research project is intended to prepare for a future longitudinal study to analyse the effectiveness of this system as anti-phishing support tool. This approach is modelled after longitudinal studies conducted by anti-phishing approaches into training users such as NoPhish [12].

8.3.1 Research Questions

There are multiple planned research questions intended to be answered as part of this study:

- Evaluating the system performance of the CatchPhish system over a longer time period
- Understanding how users interact with malicious URLs and phishing
- Analysing typical user browsing habits
- Collecting data on the make-up of URLs themselves
- Evaluating how effective the CatchPhish system is as a support aid to users for reducing phishing

Each of these research questions are possible to evaluate given the richness of the data collected by the CatchPhish system (with appropriate accounting for the Hawthorne effect [45]). A selection of automated analysis tools have also been implemented to aid the analysis of these outcomes for use in the future study.

8.3.2 Preparation

Various implementation decisions have been made to prepare for such as study. This is the primary motivation of the work in the Chapter 6. Specifically, the work done to capture user usage patterns, the implementation of data controls and the addition of privacy improvements for users of the extension.

While the study has not been run, in part do to the COVID situation, I did create a study plan which I detail below.

8.3.3 Methodology

To understand what users have learned as a result of the tool, pre- and post-testing is to be conducted to measure users' understanding of phishing, URLs and specific malicious indicators as part of a lab study. The study is intended to follow a similar format to the interviews conducted for requirements gathering in the previous year (see Section 2.1.1). The post-testing is planned to be complemented with an interview section exploring users' experience with the tool - to identify potential improvements.

Analysis of learning outcomes may be supplemented by delayed post-testing after a few months. This would be to evaluate if this system has improved phishing knowledge retention rates over traditional training approaches, an intended advantage of this tool. This is an aspect of the study which requires further consideration, due to questions around the ease of comparison between this system and traditional approaches.[3]

8.3.3.1 Participants and System Usage

There are 10 to 15 participants planned for the study, with the study planned to last for around two weeks. Due to the invasiveness of the data gathering involved, the intention is to financially remunerate each participant. These users are to be sourced from a participant pool of above-average technical skills, to match the target audience of the tool.

Participants themselves, after pre-testing, will be given support to install the tool onto any Chrome browsers on any non-mobile devices they actively use.[4] They will then be expected to freely use the tool during their daily browsing activities. Throughout this time, the tool will be supporting users through passive warnings, raising active warnings if they visit a malicious site and providing contextual information on any site a user wishes to analyse throughout this time.

[3]CatchPhish is intended to be continuously used - reducing the need for high user knowledge retention rates.

[4]There is currently no support for Chrome browser extensions on mobile platforms.

8.3.3.2 Collected Data

The system itself will be gathering URL browsing information, analysing each of the URLs processed by the tool and gathering the users' tool usage patterns: for instance, whether they click through an active warning to a malicious site. Concurrent to this, the Analysis Server will be logging performance measurements such as server load through built-in features of the App Engine platform.

One ongoing consideration is how often users are likely to see phishing emails/URLs in their typical routines. This is something this study hopes to identify as a secondary research outcome. This outcome must be balanced against understanding how the tool influences users' interaction with malicious URLs. One possible outcome is if users do not naturally encounter malicious URLs whilst using the tool. This would limit our ability to analyse the effectiveness of the tool.

Therefore, one considered option is to simulate user encounters with malicious links by sending them some mock phishing emails during the study. Since the tool analyses all URLs on any site, browser based email-clients are also flagged and updated. These mock emails can be filtered out of analysis of typical user patterns but still provide a useful way to understand how users interact with the tool when presented with phishing.[5]

The results of the study will then be analysed relative to each participants user patterns. At the end of the study, after post analysis one consideration was to present users with an infographic of their own usage patterns to supplement their personal learning from the study.

8.4 Summary

This chapter outlines the studies used to evaluate the final system. It describes the methodology of the URL analysis study, used to evaluate the accuracy of the system. It also outlines the results of the expert evaluation of the extension's UI components, conducted with an expert in security and secure interface design. The results of this study commend the tool as having a high usability standard. Alongside both of these studies, the preparations involved in the future longitudinal study are also discussed.

[5]The URLs included in the emails need not be malicious, instead amusing benign URLs such as this can be used - with the URLs true destination potentially camouflaged by the system.

Chapter 9

Results and Discussion

This chapter discusses the results from the URL Analysis Study, which are used to evaluate both the system and the algorithm. There is also a discussion of the ethics and security implications of deploying such a system for research purposes.

The goal of the URL Analysis Study was to evaluate both the algorithmic and integrated performance of the system. Therefore, the focus of this analysis is on evaluating how the system might perform in a deployed context, as an analysis server for a user-facing tool. The results of this study are analysed to provide an overview of how the system might perform in both artificial and more realistic settings.

9.1 Algorithm Performance

The algorithmic performance of the system is evaluated using four datasets. These datasets (Figure 8.1) are used to analyse the performance of the system for different settings. As discussed in Section 7.3, the use of Datasets A and B for evaluating the algorithm performance sets a particularly high bar for this evaluation.

However, whilst the ParaCrawl data (Dataset A) is atypical of what users might expect to see in a real world setting, the URLs in the PhishBank dataset (Dataset B) are representative of the phishing URLs users might expect to see (albeit typical users would expect to see much smaller proportions of phishing rather than safe URLs). As these Phishbank URLs are from an archived source, the system's performance on these URLs represents how the system might perform for URLs that are not present in the system's utilised blacklists. This therefore helps illustrate how the system might perform when identifying phishing URLs used in zero-day attacks (although these URLs will naturally have earlier creation times than those sourced from Phishbank).

The lack of representativeness of the ParaCrawl data is compensated by analysis of the more realistic datasets in Section 9.1.2. As the URLs sourced from Reddit (Dataset C) are highly popular in nature, this represents URLs that users are more likely to encounter. This is because users on the whole tend to visit a small amount of sites with a high frequency [9]. This dataset is complemented by the inclusion of data drawn

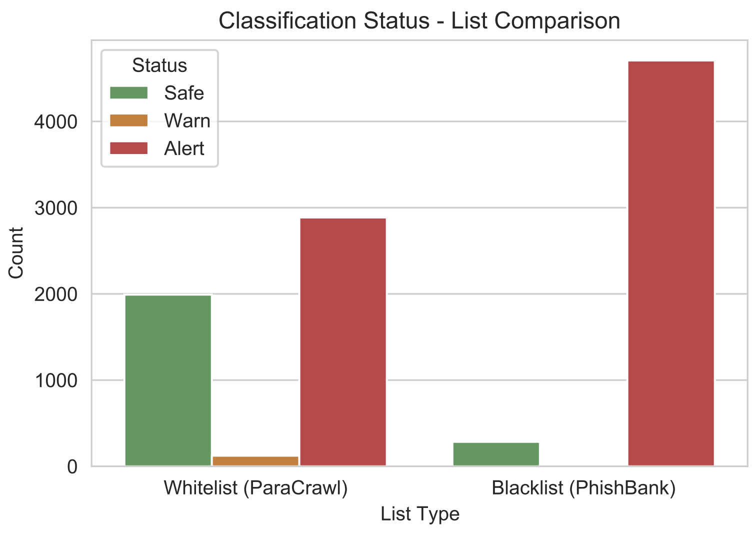


Figure 9.1: URL classification status is illustrated for both Datasets A and B, displaying the occurrences of the final URL classification categories.

from a real user, discussed in Section 4.2.1, which presents the most realistic context for evaluating the typical performance of the system.

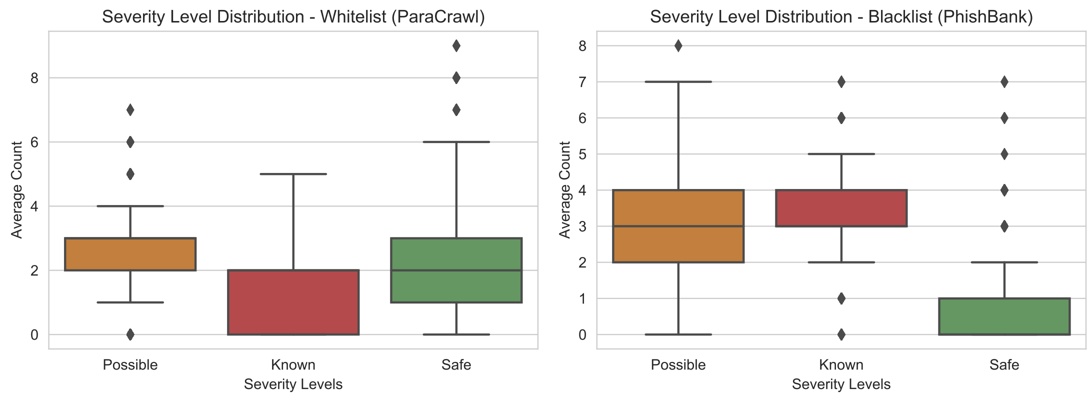
The results of the algorithm on these datasets therefore present a representative idea of how this algorithm might perform in a deployed environment.

9.1.1 Artificial Datasets

As illustrated by Figure 9.1, the additional improvements made to the system - the addition of the categorised data and the use of Fortiguard data as a safety metric - helped to improve the false negative rate of the system (when classifying the safe but less globally popular URLs of Dataset A). The addition of these metrics also had a limited impact on the true positive rate, as illustrated by the classifications of malicious URLs. There are also improvements in the amount of 'known' classifications which have occurred for whitelists as illustrated by Figure 9.2a. The positive effect of the addition of the categorised list data is illustrated in Figure 9.3.

However, Figure 9.1 illustrates that the the system still has a considerably high false negative rate when processing URLs sourced from Dataset A. The results also highlight an increased occurrence of 'safe' activations for individual heuristics on Dataset B. This demonstrates a limited impact on the false positive rate, albeit a small one, due to the changes made to improve the false negative rate.

When analysing the results of Datasets A and B according to their individual metrics, Figure 9.3 and Figure 9.4 highlight the impact of the work in Section 7.3.3 to help improve the robustness of the algorithm. The system no longer has issues with heuristics such as Typosquatting and Different TLD, with the system analysis being in line with research expectations. This robustness illustrates the utility of the system as an



(a) Dataset A: ParaCrawl (b) Dataset B: PhishBank

Figure 9.2: These graphs represent the average count/occurrence for each severity level across the aggregated heuristics. These severity level distributions are presented for both Datasets A and B.

research tool which is able to produce reliable analysis results for any desired URL dataset.

When analysing the reasons for why URLs are classified in more detail we can see a similar pattern as the prior results in Section 7.3. Figure 9.5 demonstrates the polarisation of URL classifications: either marked as safe by the safety algorithm or being marked as alert by the primary classification algorithm.

9.1.2 Realistic Datasets

As illustrated by Figure 9.6, the system performs well on the most representative and realistic datasets. The Reddit Popular Domains data is overwhelmingly classified as safe, with only a few URLs are missed classified as malicious.

In addition to improved classification of this system on the URLs, we see that the the distribution of severity level activations is more in line with what we would expect to see for whitelisted URLs (in contrast to the results in Figure 9.2a).

When we break down the individual results of these URLs by individual metric, we can see the greater impact of the popularity safety metrics. This highlights the effectiveness of the system for classifying popular URLs.

When looking at the reasons for the classification of the URL categories for both datasets, we can see that the URLs with safe classifications tend to be marked as safe in the initial stages of the algorithm.

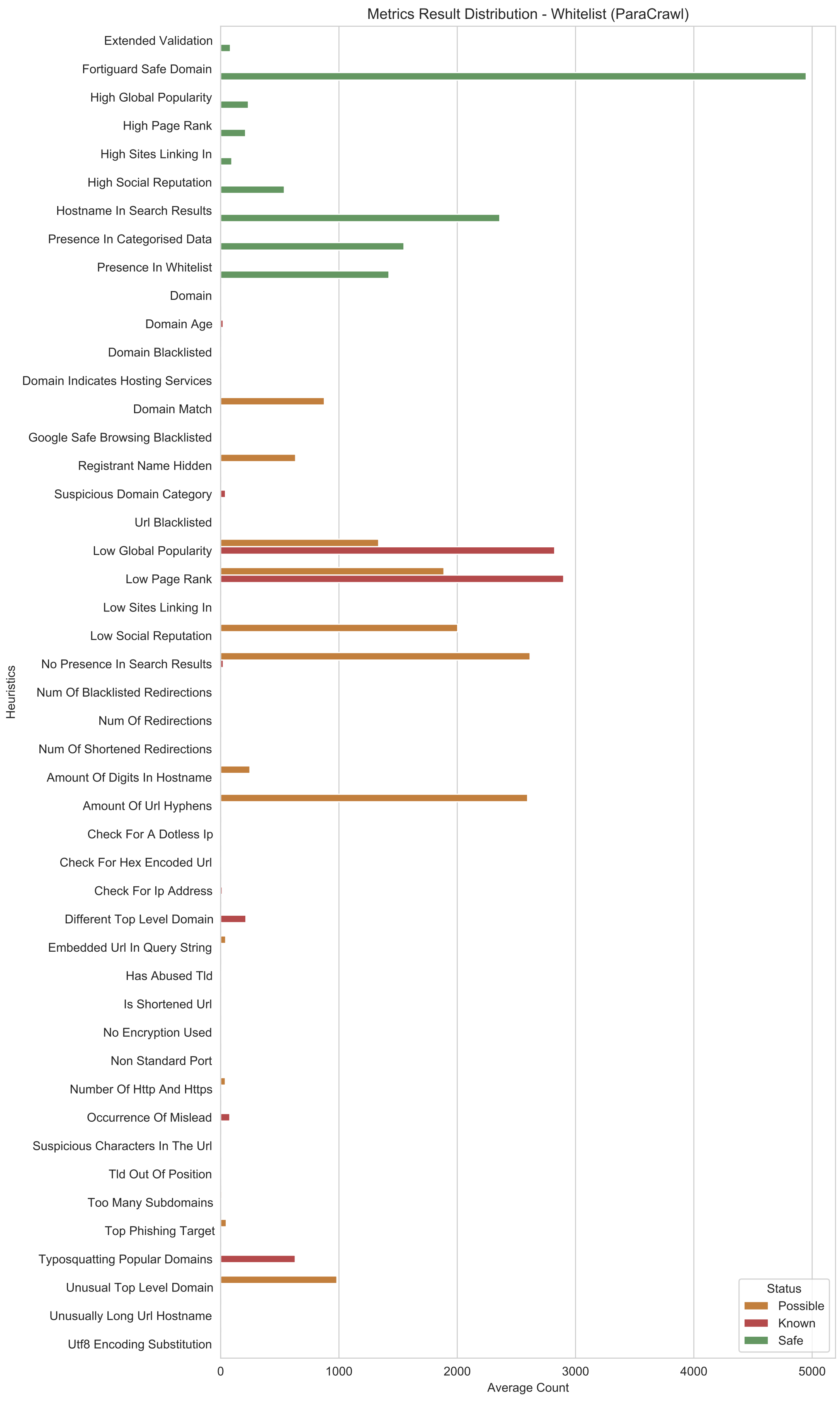


Figure 9.3: Dataset A results highlighting the performance of individual heuristics. Each heuristic is illustrated with the number of times it activated with it's associated severity levels.

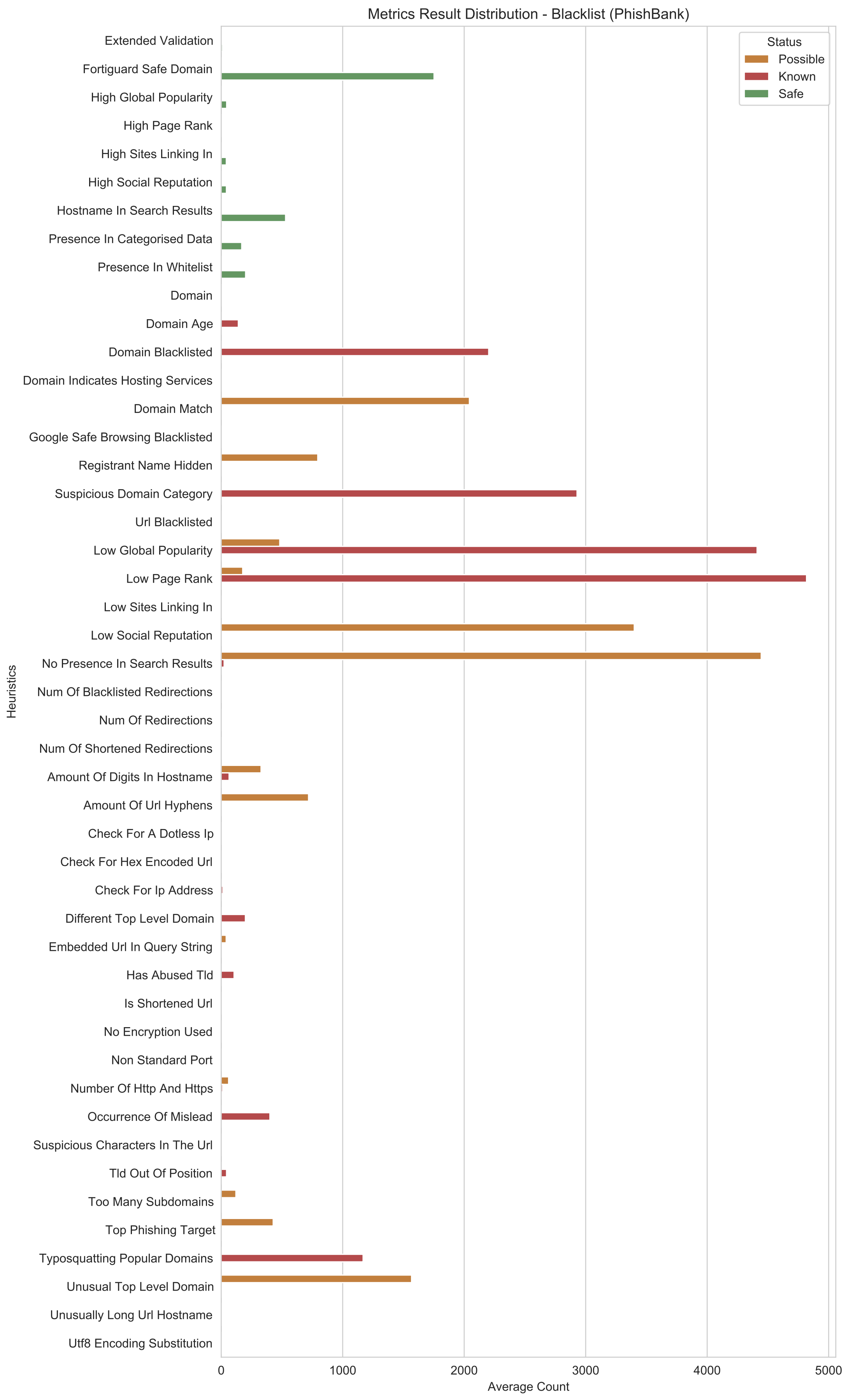


Figure 9.4: Dataset B results highlighting the performance of individual heuristics. Each heuristic is illustrated with the number of times it activated with it's associated severity levels.

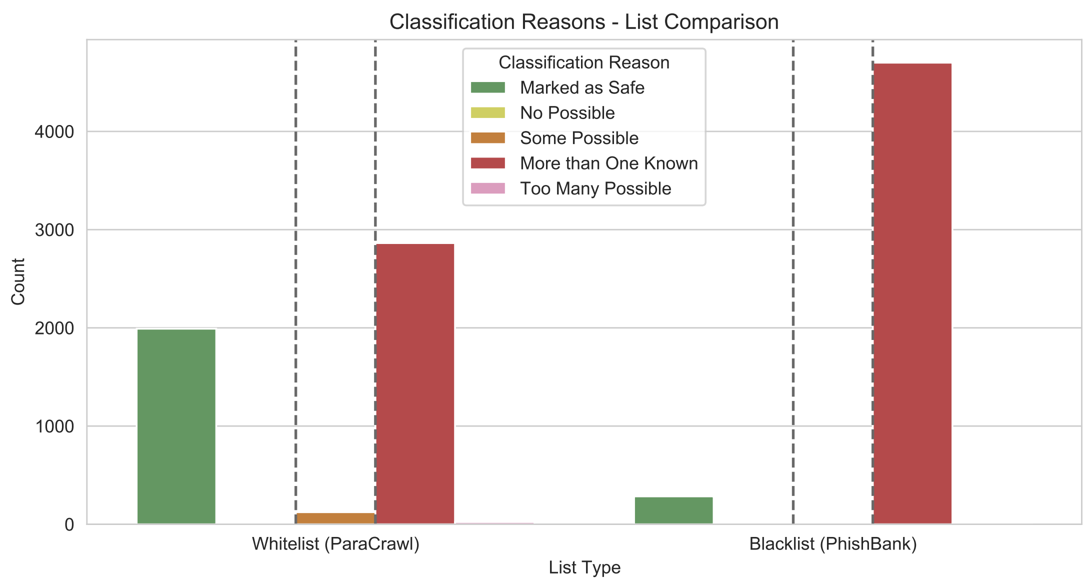


Figure 9.5: The URLs from Datasets A and B are grouped by how they have been classified, represented by their colour, the URLs are further grouped by the reason for their classification and contrasted with their occurrence count.

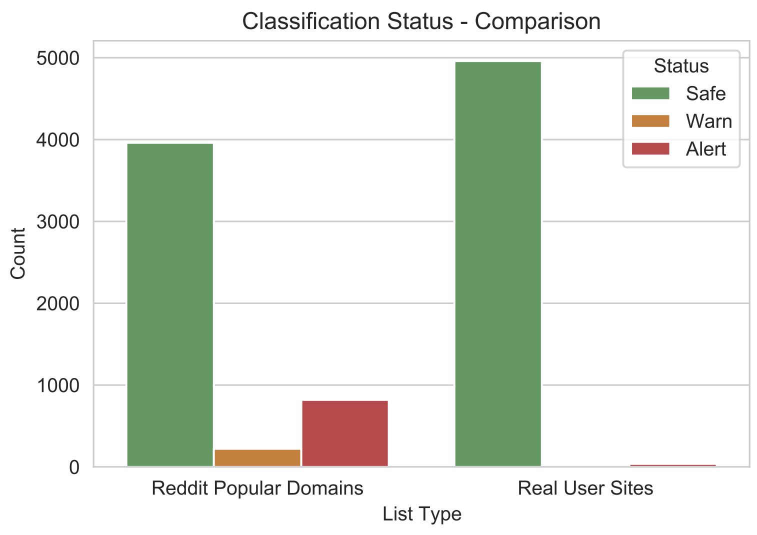
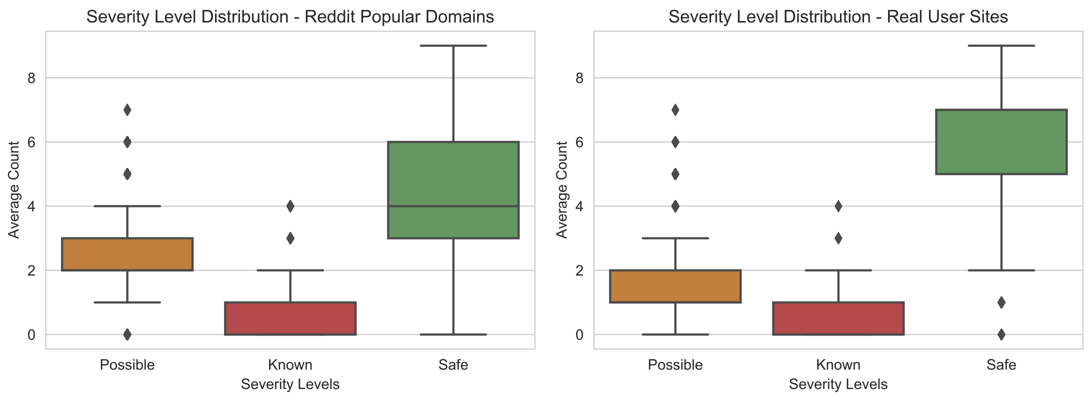


Figure 9.6: URL classification status is illustrated for both Datasets C and D, displaying the occurrences of the final URL classification categories.



(a) Dataset C: Reddit Popular Domains

(b) Dataset D: Real User Data

Figure 9.7: These graphs represent the average count/occurrence for each severity level across the aggregated heuristics. These severity level distributions are presented for both Datasets C and D.

9.1.3 General Algorithmic Performance

On the whole, the system performs well on datasets representing its intended use cases: the high amount of popular and safe URLs typically visited by users, and also malign phishing URLs. Whilst the system does not achieve the performance seen in alternative works using automated phishing detection approaches [10], this is not the primary intention of the system. It is intended to be a support system for training users rather than a detection tool. Its primary purpose is to use the heuristics for educational purposes rather than classification. The tool's ability to accurately classify URLs is an added bonus, a feature which helps facilitate user training. The system in reality is only intended to detect malicious URLs were existing systems have failed.

When analysing the results of the system in greater detail, we can see the impact of individual heuristics on a more micro level. The results illustrate the greater effectiveness of some heuristics over others for classifying the URLs. For example, the popularity heuristics, present in both the safety metrics and page heuristics, for classifying popular and malicious URLs respectively.

There are also some peculiarities with heuristics such as the amount of hyphens in URLs. The activation occurrence of this heuristic is varied across all of the results. Whilst this heuristic has a basis as a phishing feature, particularly when compared to heuristics which measure the sole occurrence of a hyphen (Section 7.2), the results appear to suggest this is a further example of a heuristic affected by URL syntax trends. The effectiveness of heuristics such as these leaves room for the wider set of heuristics to be further refined in future work.

The analysis of the individual heuristics on these datasets illustrates the utility of this system as a research platform. The system's best qualities are represented by it's ability to allow researchers the flexibility to trial various heuristic approaches. This therefore allows the system to continue to improve on its algorithmic performance over time, as intended.

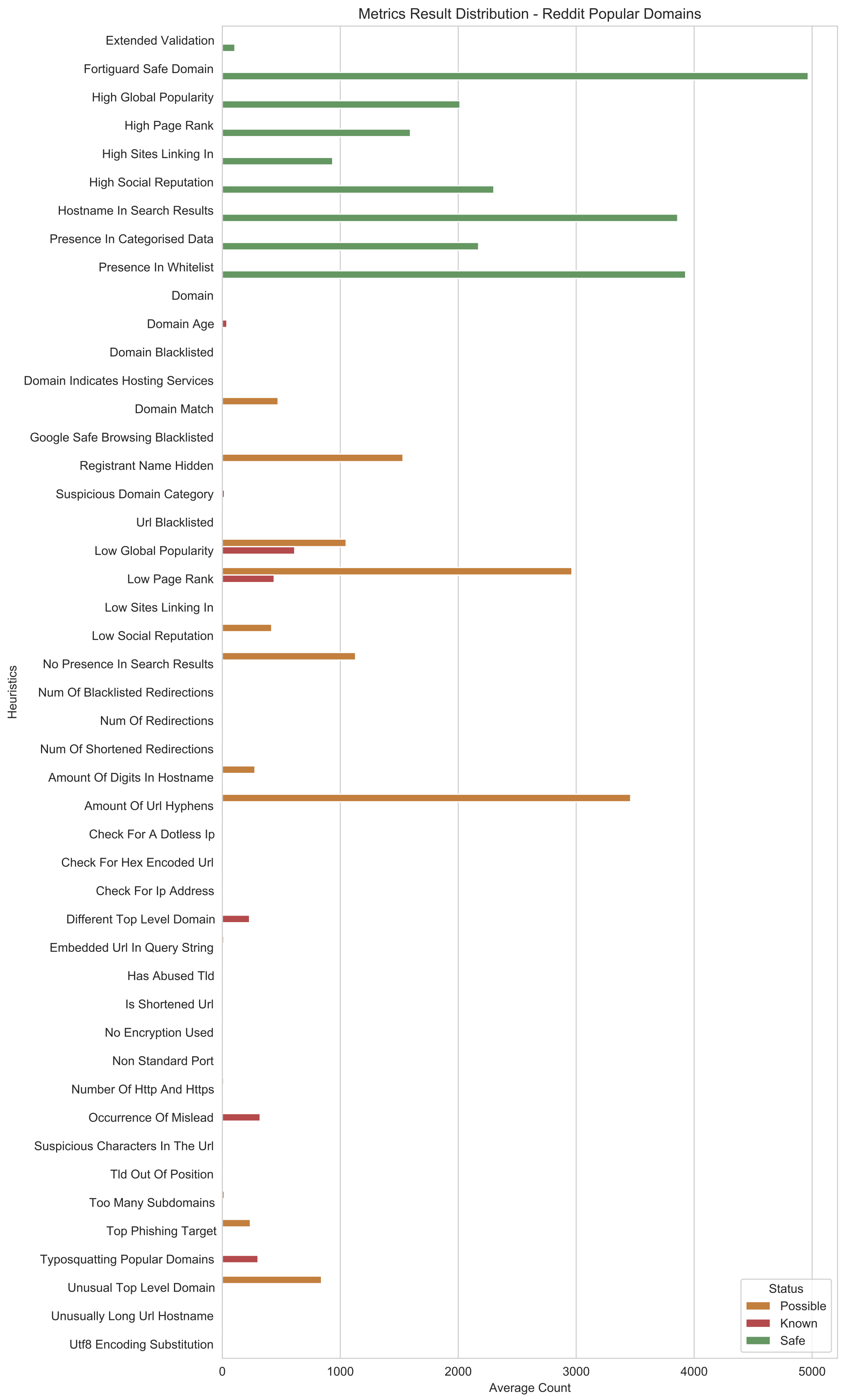


Figure 9.8: Dataset C results highlighting the performance of individual heuristics. Each heuristic is displayed with the number of times it activated with it’s associated severity levels.

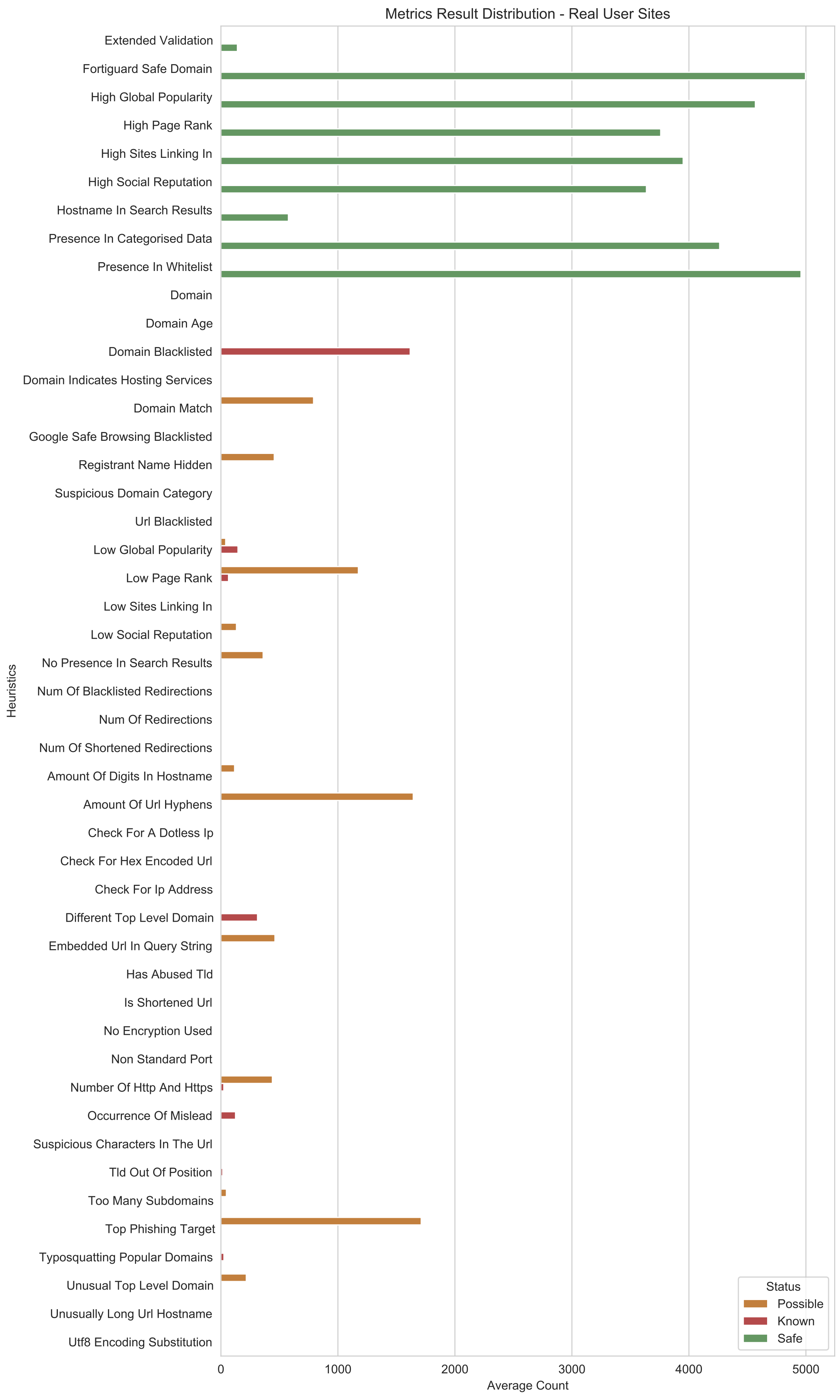


Figure 9.9: Dataset D results highlighting the performance of individual heuristics. Each heuristic is displayed with the number of times it activated with it's associated severity levels.

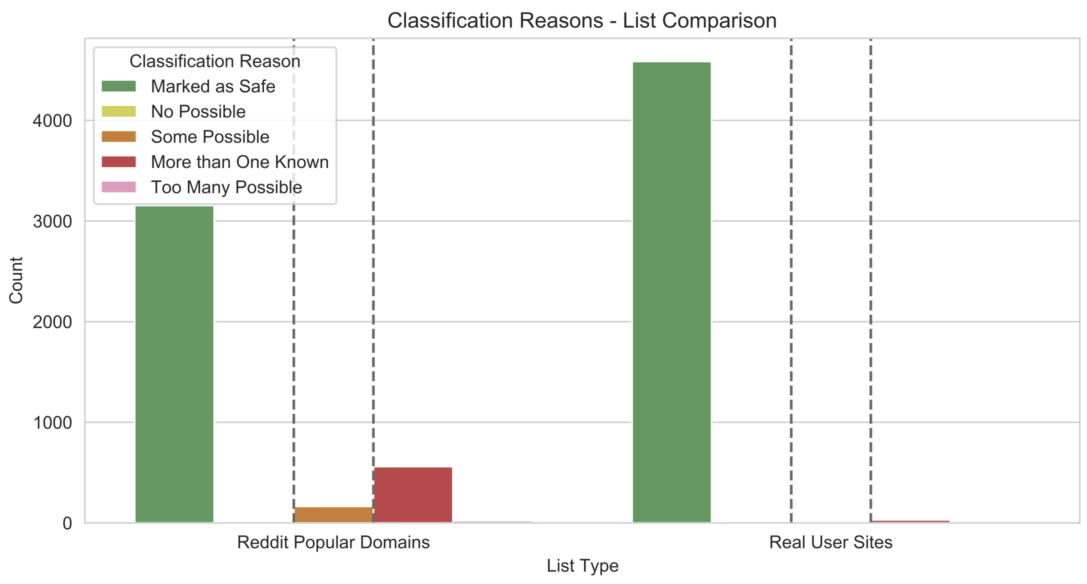


Figure 9.10: The URLs from Datasets C and D are grouped by how they have been classified, represented by their colour represented by their colour, the URLs are further grouped by the reason for their classification.

Generally, the data analysis abilities of the system also highlight another potential use case - the ability to lexically analyse every aspect of URLs. This can be achieved through either post-analysis of recorded data or implemented by the server in flight. This would allow the system to also produce research into what URLs generally look like, in the same vein as existing research [7].

9.2 System Response Time and Integration

The wider system metrics such as response time and the cohesion of the wider deployed system are both key metrics for understanding the effectiveness of the final work. The response time is evaluated with reference to Neilson's heuristics to understand the effect this may have on users. The wider system effectiveness is evaluated by the use of the browser extension to record the correct calculation of the UI components, whilst also demonstrating the extension's ability to analyse the make-up of web pages.

9.2.1 Response Time

The system response time was measured using Datasets C and D using the deployed analysis system on the App Engine Platform. This was intended to measure how the tool would perform in a more realistic environments. We see from the results illustrated by Figure 9.11 that the tool is able to perform to a reasonable degree - with an average response time of 11.11 seconds across both datasets (inclusive of network delay). This

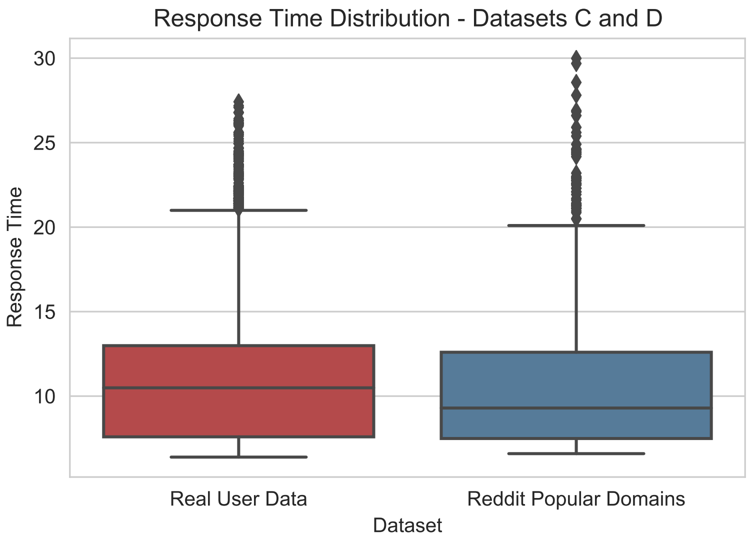


Figure 9.11: The distribution of response time results for the server when analysing Dataset D.

response time is just outwith the ideal response times Neilson's heuristics suggest is appropriate for this type of system of 10s.

However, whilst the system has a reasonable performance on these datasets, the response time analysis conducted as part of the async performance implementation (discussed in Section 7.4) suggests the system might have a less performant response time for analysing URLs in Datasets A and B.[1] However, as the datasets are more artificial in nature, this is less of concern in comparison to the general performance illustrated on Datasets C and D.

One of the factors contributing to the system response time is the complicated database queries that need to be performed. In particular, the Typosquatting heuristic. Whilst shifting the processing of this heuristic to the database allowed for a significant improvement in performance, the complicated subqueries involved in the heuristic increase the system response time. These subqueries are necessary to both calculate whether the Typosquatting has occurred and find the domain with the matching similarity score. This is to allow the detailed information to be displayed to user as part of the user statements. Should the level of information presented to users be adjusted in any way, this would allow a significant improvement in the Analysis Server response time. This demonstrates the important balance between server response times and the facilitation of user training outcomes. Ultimately, the choice made in this system is to focus on user training outcomes since this is the primary purpose of this work.

As there is a close link between the complexity of the heuristics and the response times of the Analysis Server, this will have to be closely considered in future iterations of this work. Response time improvements might still be gained whilst maintaining the training balance, by pruning the heuristics and retaining those most effective for user

[1]The differing amount of URLs from the async performance response time evaluation and the different system targets should be noted when comparing these results.

training.

Additionally, when analysing the Analysis Server response times the effect of features such as the Memorystore cache is negligible in these static evaluation contexts. This is because every URL in Dataset D is unique, reducing the positive impact of server caching. However, in a deployed setting this feature would also be expected to improve the response time of the system.

Despite the Analysis Server being slightly outwith Neilson's heuristic, the response time of the system is still well within the range of acceptability for the overall system. As previously discussed, users typically only visit a small amount of sites regularly. By adapting the browser extension cache, from its currently implemented position as a backup cache, used when users disable the extension, this can be used to improve response times for users in the general system. For instance, if the cache were to be restored to its original implementation, users would likely see response times considerably less than 10s - with the system often providing users the feeling of immediate responsiveness. The ability to get research insights into user browsing history might still be maintained by sending all user URLs to the server, but adding an additional flag to the API to indicate whether URLs should be analysed or just stored. Furthermore, since the extension itself pre-analyses URLs before users visit them, to allow the addition of the passive warnings, users are unlikely to perceive the analysis server's response time in their typical browsing experience.

9.2.2 System Integration

As discussed in Chapter 8, Selenium Testing was used to evaluate how the browser extension might perform in a deployed setting. As a result of this work, the UI Calculation and URL Processing apps were found to have a successful level of integration. For each URL's classification, the relevant user statement was produced to match each corresponding heuristic in the analysis. This was successful in all cases - highlighting the effective integration of the analysis server.

As part of this testing, we also recorded the distribution of URLs present in each site that tool was able to visit (using the URLs from the datasets). Whilst we do not use site content heuristics as part of this system due to their inherent risk, the URLs recorded on these pages illustrated some interesting patterns. The distribution of these URL is illustrated in Figure 9.12.

Figure 9.12 illustrate there are far fewer URLs present in malicious sites. This in itself may be used as an indicator of whether a site is malicious in other systems which utilise site content heuristics.

9.2.3 General System Performance

Whilst further improvements can be gained at the extension level to improve the wider system performance, the response time of the server is expected to be adequate to

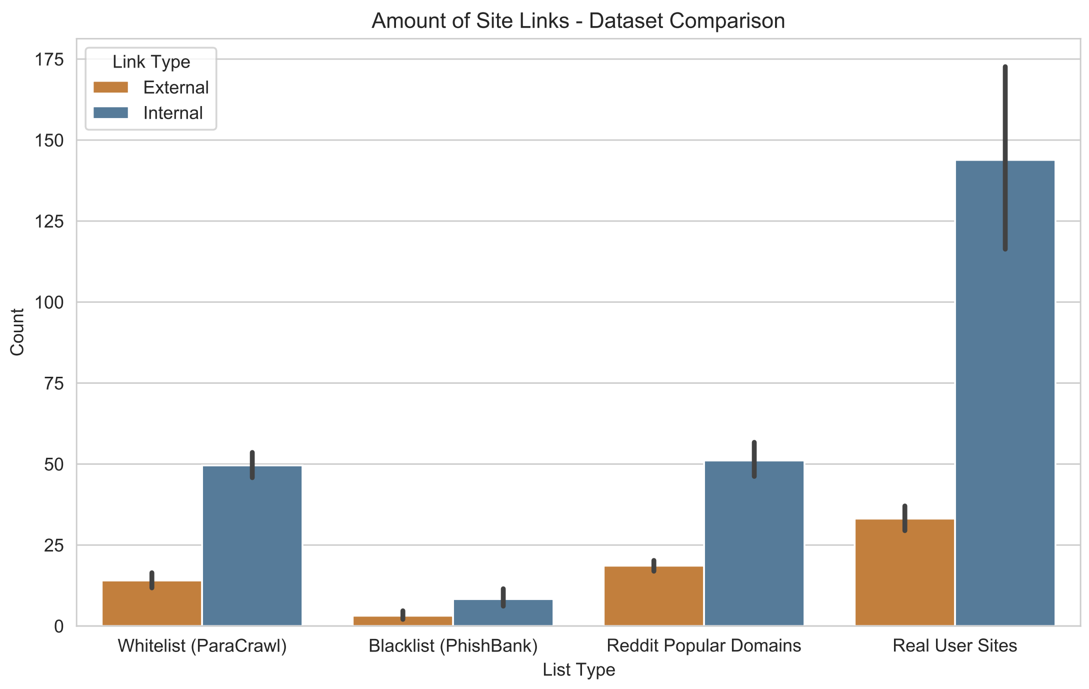


Figure 9.12: Comparison of the average count of internal and external links resulting from the browser extension's analysis of the each dataset's URLs. By extension the graph also illustrates the average total amount of URLs found on sites in these datasets.

facilitate the needs of users in a deployed setting. This is due to the limited amount of safe sites that users visit in regularity and the tool's inbuilt prefetching analysis.

The integration of the server and extension UI features is an aspect of the system which performs well on testing. The potential of the system as a general URL analysis system is also illustrated by its abilities to record substantial information about the make-up of web pages themselves.

9.3 Ethics and Security Implications

There are several ethics and security issues associated with the tool, particularly around user privacy. The tool itself gathers a significant amount of information about users: gathering each URL they visit and how they interact with the tool itself.

Even though considerations have been made in the system to allow users to be more selective about what type of data is sent (e.g. the settings option to temporarily block extension-to-server data transfer), these considerations do not wholly resolve potential problems. For instance, for users who are less privacy aware or where users have forgotten to block the transfer of sensitive data, their sensitive data will still be stored. Whilst there is some level of user anonymity in the system, pseudonymisation through user-defined non-personal IDs, even if truly anonymous IDs were to be used - user data is still susceptible to compromise.

9.3.1 An Example Security Threat

For example, whilst database roles and authentication have been set up to better secure user data, where these are not properly managed, user data may be exposed. Regardless of the anonymity of their IDs, a user's URLs contain personally identifying characteristics [32]. A user ID tied with an identifying URL can then be used by a malicious actor to collate all a users associated data.

The captured data can reveal two key attributes to a malicious actor: users usage patterns, both their times of access and sites they typically visit, and user concerns revealed by sensitive data in URLs, such as personal health data. These findings can then be used to deploy more sophisticated attacks against the user. For instance, targeted spear phishing attacks with higher likelihoods of success [34, 60].

9.3.2 Handling Privacy Concerns

Despite the existing system protection measures, there are still further features which could be introduced to improve user privacy. Foremost of these, are approaches involving the filtering of personal identifiers from URLs. There are two aspects involved in this filtering: identifying URLs with personal identifiers and redacting those elements whilst maintaining the URL.

URL personal identifier filtering is a feature which would best be implemented in the browser extension of the system - preventing any sensitive URLs being sent to the server. This ensures a user's sensitive personal data is entirely local. Thus reducing the challenges of storing sensitive data server-side, whilst reducing the level of severity of communication channel compromise.

Storing only high-level characteristics of URLs is another approach for consideration. Whilst this limits the exposure of user data, it is challenging to find the right trade-off between research outcomes and user privacy. For instance, in order to evaluate the effectiveness of the heuristics algorithm, the associated data for each heuristic needs to be stored. Rather than the full URL, only aspects of the URL such as the domain are stored. This, however, limits the future ability of researchers to analyse the generality of URLs in other studies. In addition, it is challenging to see how this approach can be successful in this system, as the volume of heuristics and affiliated data is likely to be enough to reconstruct associated URLs entirely.

The alternative system approach of employing more local processing, can be challenging from both a research and implementation perspective as well. Having users store their URLs locally limits the ability of researchers to remotely analyse collected data - increasing the complexity of scaling research studies. However, data collection issues can be tackled through user data transfers at the end of studies. This would give research participants the ability to more easily filter any data they do not wish to share (at the potential expensive of corrupting research data such as user usage patterns).

From an implementation perspective, such an approach may find it challenging to provide the richness of features implemented in the current system architecture. Other than study scalability, the use of a centralised server was used to increase analysis response times for users - by leveraging the advances of cloud computing. The challenges associated with developing browser extensions, which have limitations with library integration and alternative language support, are likely to result in fewer system features; reducing system accuracy and limiting future user enthusiasm for the tool.

For instance, one of the central features of the system is the ability to consult white and blacklists. There are two ways to implement this in a local processing approach which supports local data retention. Firstly, have a central server for hosting lists which does not store user information (with potential user information leakage through communication channel attacks), and, secondly, host the lists on each individual user's machines. A local hosting implementation might be complemented by deploying apps, such as the URL Processing app, on a user's machine - acting as local API for users' browser extensions. This latter type of approach may improve concerns with user privacy but at a significant management and performance overhead for users, likely to prevent widespread usage of the system.

9.3.3 Future Research Considerations

The ethics and privacy concerns raised by this system are sufficient enough to merit further research to seek their resolution.[2] The drawback that has to be weighted against each of the considered user privacy solutions is the reduced richness of data that can be collected for research purposes. Reducing the scope of the data collected and filtering for personal identifiers, both seem like possible ways forward. However, approaches which involve more local processing are fraught with significant implementation challenges. These more local approaches may help to secure user data for limited research studies. However, in contrast to approaches involving URL personal identifier filtering, local approaches will limit the usefulness of anti-phishing systems, preventing their development into any kind of scalable support solution to user phishing.

9.4 Summary

This chapter discusses the results of the URL Analysis Study used to evaluate the system. It places the analysis results derived from four distinct datasets in the context of how the system is intended to work in deployment. The system is found to be able to successfully analyse the URLs most likely to be encountered in intended deployment scenarios: performing well on the high amount of popular and safe URLs typically visited by users, and also malign phishing URLs.

The Analysis Server is also found to have reasonable response times on the datasets most reflective of typical user's experience with the system. The UI calculation components of the Analysis Server were also found to have integrated well with the wider CatchPhish system.

[2]Even though users can be made aware of the security implications during studies.

The chapter concludes with a consideration of the ethical and security implications of utilising this type of anti-phishing research approach. It discusses potential system security vulnerabilities and evaluates various approaches as to how these vulnerabilities might be mitigated in future iterations of this work.

Chapter 10

Conclusion

10.1 Overview

This work has focused on tackling the use of URLs as a malicious attack vector, with a particular focus on phishing. The development and improvement of the CatchPhish phishing learning and detection system has been aimed at answering the research question:

How might we develop a phishing learning and detection tool that will protect from, and inform users about, malicious URLs?

The CatchPhish browser extension developed last year was designed to be a novel contribution to anti-phishing research: combining both passive indicators, and active warnings into a tool, which actively attempts to train users - in contrast to existing work. It is through this novel continuous user training approach, which respects users as in-extractable components of phishing system loops, that improvements to anti-phishing research are expected to be gained.

After gathering requirements for the system, along with developing and evaluating the User Interface the subsequent year; the direction of this work has been to efficiently implement the analysis aspects of the system.

To do this, multiple system designs were outlined for select IaaS platforms. Comparison of these designs resulted in the selection of a Microservices-based architecture deployed on Google App Engine. This system provided the basis for the underlying anti-phishing classification algorithm. The algorithm required the processing and usage of numerous data sources to facilitate 46 distinct heuristics. This was accompanied with user-facing statements for each heuristic, written to be dynamically displayed to users - to allow users to be trained in phishing indicators, depending on the dynamic severity of each heuristic.

When designing the server, several system requirements were outlined Section 4.1 which were each able to be implemented in the final system. Critically, the requirement for sufficient user response times was adequately met. This was advanced by fine-tuning evaluations focused on identifying appropriate heuristic thresholds and the impact of performance improvements.

Multiple preparations were made as part of this work for a future longitudinal user study to identify the longer-term effectiveness of this system as an anti-phishing tool. This work included improvements to the tool's user-facing features, consideration of secure data handling and corresponding evaluations of the conducted work.

The system was also evaluated with regards to its accuracy. It was found to perform well on the high amount of popular and safe URLs typically visited by users, and also malign phishing URLs. The analysis of the URLs collected during the evaluation also illustrated interesting URL patterns among safe and malicious URLs, and highlighted the significant potential of the tool as a URL analysis platform.

10.2 Further Work

One of the main focuses of the project in the future will be the planned longitudinal study. Now that the system itself has been identified to be functionally sound, it is necessary to thoroughly evaluate the content and learning gains of the users. This study will have the additional benefit of providing a significant amount of research material for general analysis of URLs as highlighted in Section 9.1.

A critical aspect of such a system, and wider research data collection systems, is the need to answer user ethics and privacy concerns. Research into the filtering of personal identifiers from URLs should be a focus of further research to incorporate the identified privacy benefits into the CatchPhish system.

Overall, the CatchPhish system presents itself as a versatile research platform for experimenting with different URL analysis algorithms. Using this platform the effect of various heuristics and alternative approaches can be utilised to effectively further research into URL usage patterns and anti-phishing approaches.

Bibliography

[1] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek. Deploying microservice based applications with kubernetes: Experiments and lessons learned. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 970–973, 2018.

[2] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, December 1999.

[3] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 257–272, 2013.

[4] Al-Debagy and Martinek. A Comparative Review of Microservices and Monolithic Architectures. *arXiv:1905.07997 [cs]*, may 2019. arXiv: 1905.07997.

[5] Sara Albakry, Kami Vaniea, and Maria Wolters. What is this url's destination? empirical evaluation of web users' url reading. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, 12 2019.

[6] Alexa. The top 500 sites on the web. https://www.alexa.com/topsites, [Accessed March 30, 2019].

[7] Kholoud Althobaiti, Ghaidaa Rummani, and Kami Vaniea. A review of human- and computer-facing url phishing features. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 182–191. IEEE, 2019.

[8] Kholoud Althobaiti and Kami Vaniea. Is this url safe to click on? supporting users' comprehension of phishing features. In Submission, 2018.

[9] Kholoud Althobaiti, Kami Vaniea, and Serena Zheng. Faheem: Explaining urls to people using a slack bot. 2018.

[10] Simon Bell and Peter Komisarczuk. An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW 20, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Gamze Canova, Melanie Volkamer, Clemens Bergmann, and Benjamin Reinheimer. Nophish app evaluation: Lab and retention study. 2015.

[12] Gamze Canova, Melanie Volkamer, Clemens Bergmann, and Benjamin Reinheimer. NoPhish App Evaluation: Lab and Retention Study. In *Proceedings of the Workshop on Usable Security and Privacy*, 2015.

[13] Softpedia Catalin Cimpanu. Hidden javascript redirect makes phishing pages harder to detect. https://news.softpedia.com/news/hidden-javascript-redirect-makes-phishing-pages-harder-to-detect-505295.shtml, [Accessed March 30, 2019].

[14] Kang Leng Chiew, Kelvin Sheng Chek Yong, and Choon Lin Tan. A survey of phishing attacks: Their types, vectors and technical approaches. *Expert Systems with Applications*, 106:1 – 20, 2018.

[15] Cisco. What is cisco anti-spam's catch rate, false-positive rate, and throughput? https://www.cisco.com/c/en/us/support/docs/security/email-security-appliance/118198-qanda-esa-00.html, [Accessed March 27, 2019].

[16] Lorrie Faith Cranor. A framework for reasoning about the human in the loop. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, UPSEC'08, pages 1:1–1:15, Berkeley, CA, USA, 2008. USENIX Association.

[17] Adele Cutler, David Cutler, and John Stevens. *Random Forests*, volume 45, pages 157–176. 01 2011.

[18] Material Design. Material design. https://material.io/design/, [Accessed February 19, 2019].

[19] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 581–590, New York, NY, USA, 2006. ACM.

[20] Dragoni, Giallorenzo, Lafuente, Mazzara, Montesi, Mustafin, and Safina. *Microservices: Yesterday, Today, and Tomorrow*. Springer International Publishing, Cham, 2017.

[21] Let's Encrypt. Let's encrypt. https://letsencrypt.org/, [Accessed February 19, 2020].

[22] Miquel Esplà-Gomis, Mikel L. Forcada, Gema Ramírez-Sánchez, and Hieu Hoang. Paracrawl: Web-scale parallel corpora for the languages of the eu. In *MTSummit*, 2019.

[23] Medium Gil Fink. Building a chrome extension using react. https://medium.com/@gilfink/building-a-chrome-extension-using-react-c5bfe45aaf36, [Accessed February 19, 2019].

[24] Google. Chrome extension getting started tutorial. https://developer.chrome.com/extensions/getstarted, [Accessed March 27, 2019].

[25] Google. chrome.webrequest. https://developer.chrome.com/extensions/webRequest, [Accessed March 27, 2019].

[26] Google. Google. https://www.google.com/, [Accessed February 19, 2019].

[27] Google. Google chrome platform. https://developer.chrome.com/home, [Accessed January 25, 2019].

[28] Google. Google site warning. [Accessed 01/02/19].

[29] Google. Safe browsing site status. https://transparencyreport.google.com/safe-browsing/search?hl=en_GB, [Accessed March 30, 2019].

[30] Bruce Hanington and Bella Martin. *Universal methods of design: 100 ways to research complex problems, develop innovative ideas, and design effective solutions*. Rockport Publishers, 2012.

[31] A. Holmes and M. Kellogg. Automating functional tests using selenium. In *AGILE 2006 (AGILE'06)*, pages 6 pp.–275, 2006.

[32] Yuan Hong, Xiaoyun He, Jaideep Vaidya, Nabil Adam, and Vijayalakshmi Atluri. Effective anonymization of query logs. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM 09, page 14651468, New York, NY, USA, 2009. Association for Computing Machinery.

[33] Jamshidi, Pahl, Mendonca, Lewis, and Tilkov. Microservices: The Journey So Far and Challenges Ahead. *IEEE Software*, 35(03):24–35, may 2018. Place: Los Alamitos, CA, USA Publisher: IEEE Computer Society.

[34] Katharina Krombholz, Heidelinde Hobel, Markus Huber, and Edgar Weippl. Advanced social engineering attacks. *Journal of Information Security and applications*, 22:113–122, 2015.

[35] FortiGuard Labs. Fortiguard web filter categories. https://fortiguard.com/webfilter/categories, [Accessed March 19, 2020].

[36] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA, 2019. Internet Society.

[37] Samuel Marchal, Kalle Saari, Nidhi Singh, and N Asokan. Know your phish: Novel techniques for detecting phishing sites and their targets. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 323–333. IEEE, 2016.

[38] Dean Blackbourn Mark Button, David Shepherd and Martin Tunley. Annual fraud indicators 2016. Technical report, University of Portsmouth Center for Counter Fraud Studies, 2016.

[39] Manuel Mazzara and Sergio Govoni. A case study of web services orchestration. In Jean-Marie Jacquet and Gian Pietro Picco, editors, *Coordination Models and Languages*, pages 1–16, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[40] Antonio Messina, Riccardo Rizzo, Pietro Storniolo, Mario Tripiciano, and Alfonso Urso. The database-is-the-service pattern for microservice architectures.

In *International Conference on Information Technology in Bio-and Medical Informatics*, pages 223–233. Springer, 2016.

[41] Thomas Nagunwa. Behind identity theft and fraud in cyberspace: the current landscape of phishing vectors. *International Journal of Cyber-Security and Digital Forensics*, 3(1):72–84, 2014.

[42] Lily Hay Newman Netcraft. Internet security and data mining - anti-phishing. https://www.netcraft.com/, [Accessed January 25, 2019].

[43] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[44] Node.js. Node.js. https://nodejs.org/en/, [Accessed February 19, 2019].

[45] David Oswald, Fred Sherratt, and Simon Smith. Handling the hawthorne effect: The challenges surrounding a participant observer. *Review of Social Studies*, 1:53–74, 11 2014.

[46] Shari Lawrence Pfleeger, M Angela Sasse, and Adrian Furnham. From weakest link to security hero: Transforming staff security behavior. *Journal of Homeland Security and Emergency Management*, 11(4):489–510, 2014.

[47] Phishbank. Phishbank. https://phishbank.org//, [Accessed March 19, 2020].

[48] PhishTank. Phishtank. https://www.phishtank.com/, [Accessed February 19, 2020].

[49] Erika Shehan Poole, Marshini Chetty, Tom Morgan, Rebecca E Grinter, and W Keith Edwards. Computer help at home: methods and motivations for informal technical support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 739–748. ACM, 2009.

[50] proofpoint. Quarterly threat report q3 2018. Technical report, 2018.

[51] Ronald L Rardin and Reha Uzsoy. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[52] React. React. https://reactjs.org/, [Accessed February 19, 2019].

[53] E. M. Redmiles, A. R. Malone, and M. L. Mazurek. I think they're trying to tell me something: Advice sources and selection for digital security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 272–288, May 2016.

[54] Mark Richards. *Software architecture patterns*, volume 4. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA , 2015.

[55] Search Security. email spoofing. https://searchsecurity.techtarget.com/definition/email-spoofing, [Accessed March 27, 2019].

[56] Wombat security technologies. State of the phish 2019. Technical report, 2019.

[57] Lorrie Faith Cranor Serge Egelman and Jason Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. pages 1065–1074, 2008.

[58] Steve Sheng, Bryant Magnien, Ponnurangam Kumaraguru, Alessandro Acquisti, Lorrie Cranor, Jason Hong, and Elizabeth Nunge. Anti-phishing phil: The design and evaluation of a game that teaches people not to fall for phish. volume 229, pages 88–99, 01 2007.

[59] Symantec. Catch rate and effectiveness of spam caught by gateway products. https://support.symantec.com/en_US/article.TECH195964.html, [Accessed March 27, 2019].

[60] WH Symantec. Advanced persistent threats: A symantec perspective. *Symantec World Headquarters*, 2011.

[61] tetraph. Oauth 2.0 and openid, covert redirect vulnerability. http://tetraph.com/covert_redirect/oauth2_openid_covert_redirect.html, [Accessed March 27, 2019].

[62] S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.

[63] tripwire. 6 common phishing attacks and how to protect against them, the state of security. https://www.tripwire.com/state-of-security/security-awareness/6-common-phishing-attacks-and-how-to-protect-against-them/, [Accessed March 30, 2019].

[64] All About UX. Methods for expert evaluation. https://www.allaboutux.org/expert-methods, [Accessed March 28, 2019].

[65] Melanie Volkamer, Karen Renaud, Benjamin Reinheimer, and Alexandra Kunz. User experiences of torpedo: Tooltip-powered phishing email detection. *Computers & Security*, 71:100 – 113, 2017.

[66] W3Counter. Browser & platform market share. https://www.w3counter.com/globalstats.php, [Accessed March 27, 2019].

[67] Jingguo Wang, Yuan Li, and H Raghav Rao. Overconfidence in phishing email detection. *Journal of the Association for Information Systems*, 17(11):759, 2016.

[68] Rick Wash and Molly M. Cooper. Who provides phishing training?: Facts, stories, and people like me. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 492:1–492:12, New York, NY, USA, 2018. ACM.

[69] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-scale automatic classification of phishing pages. In *NDSS '10*, 2010.

[70] Whois. Whois domain lookup. https://www.whois.com/whois/, [Accessed March 30, 2019].

[71] Wikipedia. Email attachment. https://en.wikipedia.org/wiki/Email_attachment, [Accessed March 30, 2019].

[72] WIRED. Google says its ai catches 99.9 percent of gmail spam. https://www.wired.com/2015/07/google-says-ai-catches-99-9-percent-gmail-spam/, [Accessed March 27, 2019].

[73] Lily Hay Newman Wired. Google wants to kill the url. https://www.wired.com/story/google-wants-to-kill-the-url/, [Accessed January 25, 2019].

[74] Min Wu, Robert C. Miller, and Simson L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’06, pages 601–610, New York, NY, USA, 2006. ACM.

[75] Weining Yang, Aiping Xiong, Jing Chen, Robert W Proctor, and Ninghui Li. Use of phishing training to improve security warning compliance: evidence from a field experiment. In *Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp*, pages 52–61. ACM, 2017.

[76] T. Yarygina and A. H. Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20, 2018.

**

Appendix A

Phishing Heuristics Proposal

UNIVERSITY OF EDINBURGH

Design Proposal for a User-facing Phishing Learning and Detection Tool

Stephen Waddell

March 28, 2019

Contents

1 Introduction

In developing a Phishing and Learning Detection tool to be deployed onto the Google Chrome platform as an extension, I have prepared an overview proposal of the back-end functionality I believe will be needed as part of this. This proposal covers a number of different points about the functionality required to classify a URL into a a respective safety category.

1.1 Goals of the tool

The primary design goals of this tool are as follows:

Goals	**Justification**
Classifying every URL: those present in any given page, the page URL itself and any the user is directed to outside the browser	This intended to be a comprehensive solution to phishing and other security exploits which employ malicious URLs
Using the browser to prevent the user visiting any URLs that have been classified as malicious without user invention.	This intended to cut down unintentional direction to malicious URLs, caused when on platforms such as email
Presenting the information on each URL to the user in an understandable way, at appropriate points of intervention	To encourage the user to learn through embedded training: allowing users to catch URLs themselves and reduce their own danger
That the end built system includes best practice of software engineering: maximising efficiency and accessibility	That the system works effectively as and when required, such that users are not do not remove it due to efficiency concerns

Table 1.1: Justification of goals

1.2 Proposal Outline

To achieve these design goals, this proposal outlines a heuristic-based algorithm for classifying a URL into a one of three states: Safe (High likelihood of safety), Warn (Possible cause for concern) and Alert (High risk of danger with URL). Each of these form the basis of a traffic light system (Green, Amber and Red respectively) which informs the user about the relative safety of each URL. In this proposal document, each of these states will be referred to by their respective colour.

Additionally, the proposal also covers the a breakdown of each possible issue with a URL. For each of these, an explanation of the issue and a proposed method is outlined to deal with that issue. The proposal also considers how the data will be sourced to solve each possible issue.

The front-end of this tool, which is not outlined in this proposal, includes the following features:

- active intervention when a user clicks a red URL (in the form of a site warning middle man)
- annotating links on each site with a badge representing the safety colour of each URL
- a breakdown of each URL using the a research based URL report[1]

The classification of each URL is used to create the appropriate User Interface display for each URL. One of the secondary goals of the tool is to achieve as low a false positive rate as possible to maintain user engagement with the tool.

Feedback on the plans outlined in this report would be much appreciated.

2 Overall Algorithm

2.1 Decision Making Algorithm

The algorithms input will be the given URL and all required information needed to consider the classification of the URL.

The algorithm begins by considering each case where a URL might be considered purely green (undoubtedly safe). If any of these indicators, such as inclusion on a reputable whitelist, are true then the URL is classified as safe and the algorithm returns that it is a green state. Otherwise the algorithm will analyse the URL and classify the given URL as having either a yellow or red state.

To achieve this, the algorithm analyses each set of features using the metrics outlined in the subsequent chapters. Each set of features is a useful indicator of a URL's safety, which can be used to classify how likely a given URL might be phishing. These indicators are referred to as issues in this proposal, and each of these issues is categorised into one of three groups: these being known, possible and no issue (which each represent a decreasing priority level).

The features that are the focus of this proposal are:

- URL parsing features
- Domain features
- Page features

Content Features could be used in addition to this but are not favoured in this proposal due to the associated high risk of parsing the content of malicious sites.
To classify the URL a count is kept of each issue that arises from the data parsing. This is matched with threshold values to classify the URL.

Known Issues	Possible Issues	Output
>=1	>=0	Red
0	>=5	Red
0	<5	Warn
0	0	Green

Table 2.1: Thresholds for Output

The resulting output is then feed back into the system for use by the User Interface. For each result, the state of the URL is returned along with the information used to calculate that state. This information is displayed to the user in the breakdown of the URL. For those in a green state, the reason indicating its safety is returned and if it is red or yellow, the issues included in the URL are also returned.

2.2 Limitations

This approach favours limiting the number of URL's classified as yellow as much as possible. This is because one of the major benefits of the tool is the tool's ability to actively intervene to prevent the user from visiting a malicious site. This only occurs for URLs classified as the red state. At this point users benefit from embedded training as they have the break-down of the URL information displayed to them, in a real-time scenario.

To limit the number of URLs in the yellow state, the algorithm has lower thresholds for classifying the URLs as a red state. One of the key limitations of this approach is that it potentially creates a higher false positive rate for the user. Active intervention with a high false positive rate has been shown to have a reduced effectiveness over-time[3].

To resolve this problem, the intention is to implement a personal whitelist for users to maintain. Users will be able to update this whitelist with URLs when they are presented as part of this tool. This is a constant element of the UI and is particularly useful for false positive URLs classifies as a red state. After the user whitelist a site it will not be included in the red state, instead it will be given a amber state (where active intervention does not occur but the user will still be made aware they are not entirely safe). This should reduce the false positive rate over time. In user trials, the ideal outcome would be that the tool tailors itself to users common sites over time, only showing sites that are concerning and the user has never visited before. This would therefore reduce the false positive rate over time.

Known Issues	Possible Issues	Personal Whitelist	Output
>=1	>=0	False	Red
>=1	>=0	True	Warn
0	>=5	False	Red
0	>=5	True	Warn
0	<5	False	Warn
0	0	False	Green

Table 2.2: Thresholds for Output

Another suggestion to improve the false positive rate, is that the users could be allowed to toggle the threshold for possible issues being classified in the red state. In either case, to ensure the needless use of the blacklist, particularly in cases where the URL has been blacklisted, the user will be asked for additional approval. In the User Interface this may involve presenting an alert pop-up box being displayed asking them if they are sure they wish to continue.

3 Safety Metrics

It is very hard to come up with a full proof set of safety metrics. Comparatively it is easier to verifiably say a site is malicious rather than safe. There are however some indicators that are useful to suggest a site is safe.

In the proposed algorithm, there are currently two ways that a site can be designated as safe. The first is that the site has no known or possible issues: this mean it has not bee n flagged by any of the heuristics. The second is that it matches against one or more safety metrics.

3.1 Metrics

The first metric is a URL's inclusion in the list of Alexa Top sites (7.2) and is not known to be a hosting service for other sites. The Alexa Top sites is a strong indicator of the most popular sites and popularity is an indicator of safety. This is because users do not regularly visit or return to sites known to be malicious. However, some of the most popular internet sites are known to host other sites such as *wordpress.com*. Wordpress, for instance, can be used to host other sites which contain malicious content so classifying these sites as safe would be inaccurate. Therefore the Alexa Top Sites results will be filtered using known content hosting sites and other potentially malicious sites by checking their occurrences on blacklists such as hpHosts (7.1).

An additional metric is the PageRank of the website. This is based on backlinks, which are incoming links to a webpage. By taking a measure of how many quality backlinks a webpage has it indicates how popular that webpage is. This is a metric which has been used in major search engine providers in the past to indicate how high a web page should be listed in its search results.

Looking at how often the site is shared on social media is another useful metric. Sites such as Facebook and Twitter regularly include links to external sites which are shared between its users. This is an indicator of a sites popularity, as multiple shares between users suggests it is safe since they are encouraging others to visit it.

3.2 Approach

The idea is to combine each of these metrics and use appropriate thresholds to determine if a site is popular enough to be classified as safe. By applying these metrics to each website, and ensuring each page's URL is safe in the site, we can mark sites containing multiple webpages as safe without analysing each URL, provided they do not leave that domain. The idea behind this is to increase efficiency in each site. This should not pose many problems providing the metrics are applied correctly, as there is a much smaller chance of being attacked on a site when swapping between pages from distinct sites [insert reference].

4 URL Parsing Features

This focuses on parsing the URL into its distinct components and use these to pick out indicators in the URL itself which might suggest it is malicious.

Processing each of these heuristics typically involves using natural language tools such as regex to match the contents of each URL against any concerning flags. These often involve the request of some data in order to to complete each check accurately; the limited amount of required libraries means these checks can be preformed locally on a users machine. Each metric relates to a issue level based on how much of an indicator that metric is.

4.1 Manipulation Tricks

Trick	**Explanation**	**Check**	**Data Needed**	**Issue**
Too many subdomains	Can redirect user to alternative site	regex search all '.' characters and take a count of resulting array	Amount of common URL domains; The URL domain	Possible
Typosquatting Popular Domains	Targets users who incorrectly type a web address into their browser	Similarity measure between host name and popular domains	Appropriate similarity measure (7.3); list of most popular domains	Known
Unusual top level domain (Camouflage)	Top level domain is not one of the most commonly used ones	Match top level domain with a list of most popular domains	List of most popular top level domains	Possible
Digit replacement of letters	Masking the direction of the URL by replacing characters such as 'o' with similar digits such as '0'	Use regex to count the amount of digits in the hostname, and compare to the average or typical	Typical amount of numbers in a URL hostname	Possible

Table 4.1: Manipulation Tricks

Trick	Explanation	Check	Data Needed	Issue
Shortened URLs	URLs which have been shortened means users are unable to pick out key characteristics from a URL	Use the shortener services to follow or expand the URL and express this to user; express to the user and run analytics on expanded URL (if the shortened URL is safe)	Access to shortener APIs	Possible
UTF8 encoding substitution	Substituting identical looking characters from different alphabets such as English and Cyrillic	Use a language detector to check if the detected URL language matches with the user's local language	Language detector; Access to default local language of browser	Known
Use of I.P. address, hex or decimals	Use of non-standard information makes the URL harder to understand and can be used to confuse the user	Use specific regex for checking the hostname for each of these elements	IP address regex (7.3.1); Hex regex (7.3.1); Decimal Regex (7.3.1)	Known
Substituting normal characters	Characters such as 'w' can be replaced with 'vv'	Check the work similarity with any suggested search engine search replacement	API to do search engine search (7.5) and/or get replacement text	Known
Mislead	Expected company name is embedded somewhere in the URL but not destination	Search the URL for a list of the most popular company names	List of most popular company names	Known

Table 4.2: Manipulation Tricks

Trick	Explanation	Check	Data Needed	Issue
Embedded URL in query string	Open redirection is detected based on the existence of a URL in the query string	Use regex to match URL any possible the query string	Access to URL query string; URL match regex	Possible
Number of http/https	Some attackers add additional http to trick the user about the start of the URL	Search the URL using regex to get a count of each occurrence	Regex to match protocols across the URL	Possible
Suggestive word tokens	Some words only exist in phishing URLs such as "confirm", "banking", "account", and "signin"	Search using regex for any words in a list of known suggestive word tokens	List of suggestive word tokens	Known
Suspicious characters in the URL	The existence of 'at' (@) means criminals use this to mislead users or characters such as hyphen (-)	Use of regex to check for any suspicious characters in the URL	list of suspicious characters and their occurrences [4]; regex to check whole URL	Known
Encryption	HTTPS connection is more secure. Incompetent feature but useful for safety	Check the protocol is https or http through string comparison	access to the URL protocol;	Known
Non-standard port	Whether the port belongs to a standard HTTP ports: 80, 8080, 21, 4143, 70 and 1080	Match the port type to a given port in the URL and check the protocol and the port match	list of common ports; access the URL port; access the URL protocol	Known

Table 4.3: Manipulation Tricks

Trick	Explanation	Check	Data Needed	Issue
Use of atypical deliminator character	'.' is the typical deliminator character, example is a 'home-depot.com' rather than 'homede-pot.com'	Search for deliminator characters other than '.' using regex	list of deliminator characters; regex to search for these	Possible
Unusually long URL hostname	URL hostnames that are overly long can be used to mask the true destination of the URL	Count the amount of characters in the URL, and compare to the typical URL length	Typical length of a URL hostname[2]	Possible
Different Top Level Domain (TLD)	The URL has the same domain as a popular site but a different top level domain	Compare both the popular TLD with the URL	Popular domain list; hostname of popular domain; top level domain of each	Known
TLD out of position	The appearance of a popular TLD such as "com" in the subdomain deludes users into believing this is the end of the hostname.	Check if the URL subdomain includes any popular subdomains using regex	List of popular TLDs; regex to match TLDs in subdomain; access to the URL subdomain	Known

Table 4.4: Manipulation Tricks

5 Domain Features

Domain features focus on detecting phishing domain names, such as by checking the details of the domain's registration status. In doing passive queries related to the domain name, we can detect indicators based on the known trends of malicious URLs. To handle each of these indicators, external data using API's must be requested to reason what the status of a given URL is.

Fact	Explanation	Check	Data Needed	Issue
Domain/I.P. Blacklisted	If info is blacklisted it is unlikely to be safe	Store blacklist as part of local database: query blacklist database tables for URL presence	Stored blacklists; Chrome Database APIs	Known
Days of domain registration	Phishing sites tend to be newly created	Store WHOIS information as a database: query the database to get the creation data of the website	Current date; Threshold amount of days to check URL uptime; Database or API with WHOIS Info	Possible
Registrant name hidden	This can indicate the individual doe not wish to be found and a crime is afoot	Query the WHOIS database for the registrant name: compare the registrant name to check if it has been hidden	Get an idea of what a hidden registrant looks like	Possible
Domain match	The domain exists in the WHOIS record or the domain is in the URL matches the domain in WHOIS	Check if the domain exists in the WHOIS database by querying it	WHOIS database API	Known

Table 5.1: Domain Facts

6 Page Features

Page features use information about pages which are calculated using reputation ranking services. These are some of the most useful indicators of a URL's safety if they show a site is popular, but can equally be an indicator of phishing where a site is shown to be unpopular. In this sense, they give information about how reliable a site is.

Each of these heuristics also requires the use of a number of external data sources to be able to accurately function as a heuristic.

Fact	Explanation	Check	Data Needed	Issue
Search Results	Phishing website have short life therefore, usually they are not in the result, besides, not frequent in the results	Use an api to get the first x(20) [reference needed to indicate amount] and check if the result is there	API to do a search from command line; A means to parse search results	Known
Number of redirections	Phishing sites tend to be newly created	Use an api to check the amount of redirects in a given URL	API to do this redirect search; Build understanding of what output will look like	Known
Location	The physical location of the registrant usually differs from the physical ones	Use a web service to look-up the location of the corresponding service to the website; Check this location with locations that are known to host a high amount of malicious contents	List of places that are known to host more malicious content; Loo-up web service	Possible

Table 6.1: Page Facts

Fact	Explanation	Check	Data Needed	Issue
Global Popularity	The rank that Alexa assigns to domains	Get the global popularity of the site of using the Alexa Api; Check threshold to see if this is low	Alexa Global popularity api	Known
Page Rank	The relative importance of a page within other web pages	Use page rank api; Check if the page rank is high or above threshold	PageRank api	Possible
Social Reputation	The link popularity among social media users such as Twitter and Facebook	Use social reputation api; Check if social reputation is low	Social reputation api	Possible

Table 6.2: Page Facts

7 Data Sources

There are two specific goals when it comes to requesting data from various sources. The first is security and privacy: each API transmission of a plain text URL, could expose their pattern of website behaviour to potential malicious actors. Therefore one goal in collating of requisite data, is to limit this by focusing on data which can be collected and stored internally within the tool. The second goal is efficiency: if the time taken to consider each URL is too great then the response time of the tool might be too slow for the user. This can be handled in multiple ways such as asynchronous URL prepossessing. However, requesting, where possible, data such as blacklists in bulk and storing them locally in a tool accessible database, is a means to handle this from a data perspective. The advantage of this is to prevent outside awareness of user requests. This also allows the developers to store visited user URLs more securely i.e. hashed and salted.

Alongside this, using multiple sources of data is also key to prevent an over-reliance on any one particular data source and a corruption of the tool's accuracy. This could lead to information being skewed, in turn compromising user safety.

7.1 Blacklists

Google Transparency Report:
https://transparencyreport.google.com/safe-browsing/search?hl=en$_G B$
$Database - D/L$: *https://developers.google.com/safe-browsing/v4/*

PhishTank:
https://www.phishtank.com/

hpHosts:
https://hosts-file.net/
$Database - D/L$: *https://hosts-file.net/?s=Download*

7.2 Reputation Ranking

***Alexa**:*
https://www.alexa.com/topsites

***The Moz Top 500**:*
https://moz.com/top500

7.3 URL Information

***Levenshtein Distance**:*
https://en.wikipedia.org/wiki/Levenshtein$_d istance$

7.3.1 Required regex

Hex regex:
https://stackoverflow.com/questions/9221362/regular-expression-for-a-hexadecimal-number

Decimal regex:
https://stackoverflow.com/questions/11500482/regex-to-find-integers-and-decimals-in-string

I.P. address regex:
https://www.regular-expressions.info/ip.html

7.4 Domain Information

7.4.1 Top-level domains

W3Techs:
https://w3techs.com/technologies/overview/top$_{l}evel_{d}omain/all$

Lifewire:
$https://www.lifewire.com/most-common-tlds-internet-domain-extensions-817511$

7.4.2 WHOIS Database

WHOIS Database API:
https://www.whoisxmlapi.com/

7.5 Page Information

Search Engine Spell-check:
https://azure.microsoft.com/en-gb/services/cognitive-services/spell-check/

Google search command line:
https://www.npmjs.com/package/node-googler

URL Redirect Checker:
https://httpstatus.io/help

Check Host Net:
https://check-host.net/ip-info?host=www.google.com

7.5.1 Certificate Validation

Certificate Info*:*
https://chrome.google.com/webstore/detail/certificate-info/jhldepncoippkjgjkmambfglddmjdmaj
The API behind this extension is possible to use. It only uses basic certificate validation however.

REFERENCES

[1] *K. Althobaiti and K. Vaniea. Is this url safe to click on? supporting users' comprehension of phishing features. "[Unpublished]", 2018.*

[2] *S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measurement of phishing attacks. In* Proceedings of the 2007 ACM Workshop on Recurring Malcode*, WORM '07, pages 1–8, New York, NY, USA, 2007. ACM.*

[3] *M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In* Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 601–610, New York, NY, USA, 2006. ACM.*

[4] *G. Xiang, J. Hong, C. P. Rose, and L. Cranor. Cantina+: A feature-rich machine learning framework for detecting phishing web sites.* ACM Trans. Inf. Syst. Secur.*, 14(2):21:1–21:28, Sept. 2011.*

Appendix B

Cloud Server System Designs

UNIVERSITY OF EDINBURGH
TULIPS RESEARCH GROUP

CatchPhish Server Design Proposal

Stephen Waddell

October 25, 2019

Contents

1 System Requirements

CatchPhish is a URL analysis tool intended to leverage advances in Usable Security to reduce phishing. One component of this system is a real-time phishing detection system. This detection system consists of several necessary features. Alongside these features, there are a number of requirements for how the system should operate. A longitudinal study is planned to determine the viability of the system and evaluate research outcomes.

1.1 System Features

The core features of the system are intended to be:

- An event-triggered URL analysis pipeline
- Storage of URL analysis along with user usage statistics
- Automated data analysis for researchers
- Scheduled updates of preprocessed data for the URL analysis functionality

The system should also be adaptable for the possible addition of future machine learning pipelines - either utilising the analysis data or built into the URL analysis pipeline.

1.2 Requirements

- **Processing Speed**: A URL analysis response time between 0.1 and 1 seconds[1]
- **Data Storage**: Ability to store and access system data without a significant impact on system performance, alongside allowing access to longer term TULIPS lab[2] resources. Data structures for quick lookups on natural language data are also required
- **Allows for Async Preprocessing**: The periodic population of database black and white lists should have a minimal performance impact on core system performance
- **Language Flexibility**: To make use of the best libraries for processing URLs across various languages
- **Authorisation**: Be able to prioritise requests from different sources (Chrome extension over web), being able to distinguish between these
- **Data Security**: The security of user data is an utmost priority - secure communication between all stakeholders and the server, along with an authorisation system for database access

Further optional requirements would be to store prior versions of the URL analysis by timestamp, along with the ability to process heuristics and their data sources concurrently to improve response times.

[1]https://www.nngroup.com/articles/response-times-3-important-limits/
[2]https://groups.inf.ed.ac.uk/tulips/

1.3 FRONTEND/MODELLING REQUESTS

The primary frontend for the application will be the Chrome extension[3]. This extracts all the URLs from each page a user visits. On web pages there is an average of 82 anchor tags [4] with users estimated to visit around 78 websites a day [5]. Therefore a high performance is required for processing each URL. In the testing period, at any given time there may be 10-15 users using the tool through the extension.

An additional requirement is the ability to serve a web page, which will be a secondary entry point for the analysis pipeline. The purpose of the web page will largely be to advertise the project, but there will also be the ability for users to process URLs that they can submit through this site. This will require server side rate limiting to prevent Denial of Service attacks. The amount of resources for processing website requests is of secondary importance to the requests received from the Chrome extension extension.

The server should also allow for requests from command line interfaces for testing and automation. The server should also rate limit requests from this source.

[3]https://developer.chrome.com/extensions

[4]https://www.advancedwebranking.com/html/

[5]An approximate value based on collected usage data from a small sample of users

2 EC2 System Implementation

An overview of the current system implementation.

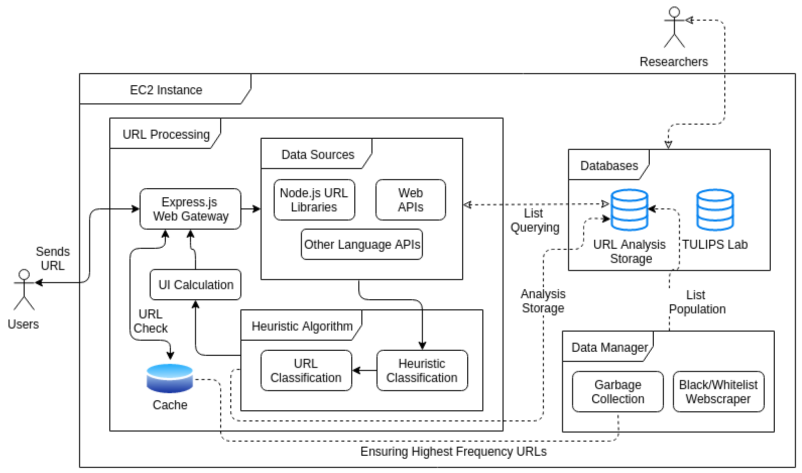


Figure 2.1: EC2 System Diagram

The current stack utilises the Express web framework [6] to facilitate a simple REST API. Node[7] URL libraries, along with additional language libraries and Web APIs are utilised for the URL preprocessing. PostgreSQL[8] is the Relational Database Management System (RDMS) utilised to store the URL analysis and TULIPs lab resources. Database authentication is managed using PostgreSQL user accounts, with authorisation of Chrome extension users utilising OAuth2[9]. Data preprocessing and web scraping is implemented in Python, with further scripts employed to automate data analytics. The system is hosted on an EC2[10] instance.

2.1 Core Application (Event-driven)

- **Web Gateway**: The gateway triggers different system functionality based on the URL path - primary functions are URL analysis and usage information storage
- **Data Sources**: Utilises Node libraries for parsing URLs along with requesting data from Web based APIs with the potential of requiring access of other language APIs
- **Heuristic Algorithm**: Heuristics process the data and utilise vulnerability thresholds to produce an individual classification for each heuristic. These are aggregated to produce an overall URL classification. One of the heuristics requires the prepopulation of

[6]https://expressjs.com
[7]https://nodejs.org/en/
[8]https://www.postgresql.org/
[9]https://oauth.net/2/
[10]https://aws.amazon.com/ec2/

a trie[11] data structure.

- **UI Calculation**: Processes some additional statements to be displayed on the UI - optional output based on the request source

2.2 Data Storage (Event-driven with Independent Accesses)

- **Cache**: Intended to increase access times by storing a limited size selection of highly frequent URLs
- **Databases**: Two PostgreSQL databases contained in the same management system. The TULIPs Lab database is not part of the application, hosting independent lab phishing resources. The URL Analysis Storage database contains the black and white lists, the processed URLs and the user usage history.
  Further Requirements: Both the Lab and the URL Analysis Storage databases need to be accessible to both researchers and automated data analysis scripts. Automated database backup is required.

2.3 System Management (Async Tasks)

- **Data Manager (Async flexible scheduling)**: Database (black and white list) population done by scheduled web scraping which are adjusted to suit server load. Also manages the application cache, ensuring it is populated with the most frequent URLs
- **Automated Data Analysis (Async periodic)**: Scripts to download and process the latest usage information and system analysis information
- **System Security**: The use of the TLS encryption protocol for server communication, and OAuth2 for authorisation to distinguish the requests of Chrome Extension users.

[11] https://pypi.org/project/pygtrie/

3 AWS Serverless Proposal

An overview of the AWS System Implementation.

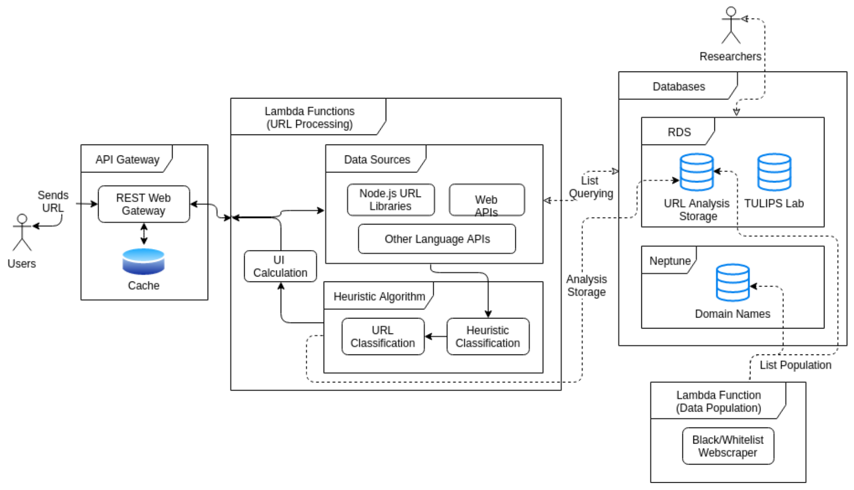


Figure 3.1: AWS Serverless System Diagram

This system design maintains the same system functionality. It splits the system components integrated in the EC2 server design into interconnected subsystems utilising separate AWS products.

3.1 API Gateway

API Gateway[12] would replace the Express.js Web Gateway. Since this manages "traffic to existing back-end systems" it will reduce DOS attacks by “throttling API call spikes” and also replace the cache feature of the application by “enabling result caching”.

The API caching works with a "specified time-to-live (TTL) period"[13] which has a maximum of 3600 seconds. The maximum size of response is 1048576 bytes.

HTTPS endpoints can be setup with the API Gateway in the same way as the Express.js server[14]. SSL certificates can be generated using the API Gateway, reducing the need to use LetsEncrypt[15] certificates.

The existing domain can be transferred[16] over to AWS using Amazon Route 53[17] (a Domain

[12]https://aws.amazon.com/api-gateway/
[13]https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-caching.html
[14]https://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started-client-side-ssl-authentication.html#configure-api
[15]https://letsencrypt.org/
[16]https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/domain-transfer-to-route-53.html
[17]https://aws.amazon.com/route53/

Name System web service). There is a cost attached to transferring domains.

Website resources including HTML and Javascript files can be stored on Amazon S3 [18].

3.2 Lambda Functions

AWS Lamda[19] is a serverless compute service which allows code to be run based on event-based triggers. This service encapsulates applications which contain various functions which can be individually triggered.

A Lambda function[20] could implement the data sources, event-driven database access and heuristic algorithm processing. Lambda functions will also be able to implement optional logic, by only triggering other Lambda functions if they meet the logic requirements (i.e. processing an additional URL on detection of a shortened link). The Lambda Function implementation is split into Applications which each have multiple Functions triggered by distinct events and do the processing using Layers. AWS Lambda Layers[21] provide additional language support by allowing Lambda Functions to pull in additional libraries written in different languages and/or run-time environments.

The Functions are called by discrete trigger events which can be based on the URL path. Code can be shared between multiple functions using Lambda Layers [22].

A separate Lambda application would be needed to allow for the data population feature (black and white list preprocessing). Lambda Functions can be triggered by inbuilt CRON jobs, which would allow for the necessary independent scheduling[23]. There is no impact on server performance by performing these jobs in a Serverless architecture unlike with deployment on EC2.

3.3 Databases

3.3.1 RDS

RDS[24] also can utilise PostgreSQL so can continue to use the same database structure. Accesses to this database access string and use as is from there. Same database separation (between URL Analysis and the TULIPS Lab databases) here as in the EC2 system design.

Automated and encrypted database backups are a feature of RDS database.[25]

[18]https://aws.amazon.com/s3/

[19]https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[20]https://docs.aws.amazon.com/lambda/latest/dg/lambda-functions.html

[21]https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html

[22]https://aws.amazon.com/blogs/aws/new-for-aws-lambda-use-any-programming-language-and-share-common-components/

[23]https://docs.aws.amazon.com/lambda/latest/dg/tutorial-scheduled-events-schedule-expressions.html

[24]https://aws.amazon.com/rds/

[25]https://aws.amazon.com/rds/details/backup/

3.3.2 Neptune

Neptune[26] is a Graph database intended to hold a trie data structure populated by the data population component. This is a replacement for a trie data structure populated and held continuously in memory. Incorporating Neptune would reduce the need for memory management using a separate subsystem as may be needed in the EC2 system design.

- **Domain Names**: Allow quick access to check for misspellings in domains, the trie being populated with the most common domain names

3.4 Cognito

Cognito[27] is an AWS application security product. IAM Roles would be useful for access controls for the other lab users to the databases[28]. The API Gateway supports Cognito OAuth2 scopes [29] which will allow Chrome extension authorisation. This is managed using Cognito User Pools[30].

[26]https://aws.amazon.com/neptune/
[27]https://aws.amazon.com/cognito/
[28]https://docs.aws.amazon.com/IAM/latest/UserGuide/access_controlling.html
[29]https://aws.amazon.com/about-aws/whats-new/2017/12/amazon-api-gateway-supports-amazon-cognito-oauth2-scopes/
[30]https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html

4 Costs and Comparison

4.1 Costs

Development and server deployment is currently on an t2.micro EC2 instance which costs \$0.0116 per hour[31]. The t2.micro instance is covered for up to 750 hours per month in the free tier for the first 12 months of usage. This is sufficient for development but a more expensive EC2 instance may be required for deployment and use during the longitudinal study.

The benefit of using Serverless system design is that most service costs are only incurred for the time they are actively being used. However, the price of each service needs to be calculated individually. Further discussion on the system architecture and the amount of requests is necessary before the cost of using this service can be thoroughly evaluated.

4.2 Comparision

The AWS Serverless system design removes the need to consider machine performance by providing a horizontally scaling system architecture which abstracts individual machine performance. Compared to the EC2 system design, this will reduce the need for future development required to scale the system along with limiting the need to implement performance management. This will allow development to focus on the general research approach such as the performance of individual heuristics. It also simplifies the wider deployment of the tool in the future and potentially increases robustness of the system.

The adaptions to the existing codebase required to fit into the Serverless system design are limited, since they both utilise Node for the triggers. The difficulty with the Serverless system design is ensuring successful usage of each of the concurrent systems.

There are also some limitations to the Serverless system design. The API Gateway cache may incur more processing with its use of TTL than the EC2 system cache which is updated with the most frequent values and managed by a Garbage Collection module. Removing this feature reduces the need to monitor the cache quality. The lack of such a cache may also be unnecessary depending on the processing time of the system, and the speed of database accesses. Additionally the inability to hold values in memory with a Lambda function removes the ability to implement a trie for word processing and requires the use of the Neptune graph database.

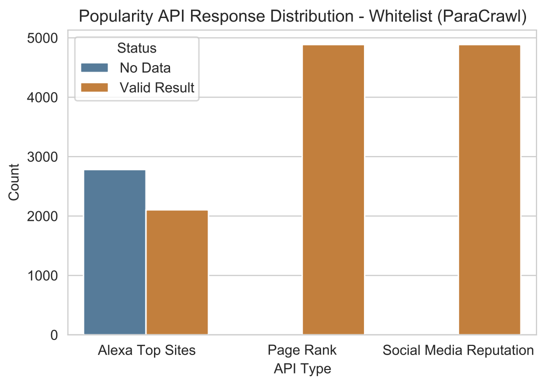
In either system there is a demand to implement URL preprocessing client side to limit user data exposure.

Due to the greater robustness in the Serverless system design, it is preferable to the EC2 system and, it should be the focus of future development.

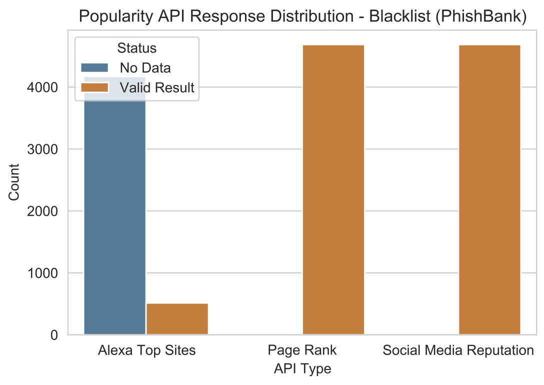
[31]https://aws.amazon.com/ec2/pricing/on-demand/

Appendix C

Iteration with Real Data - Additional Graphs



(a) Whitelist (ParaCrawl)



(b) Blacklist (PhishBank)

Figure C.1: The popularity API response distributions resulting from the Iteration with Real Data trial. These graphs highlight the initial findings in Section 7.1 - the difference between the availability of data between white and black lists for the Alexa Top Sites API. They also highlight the limited popularity data available for the ParaCrawl URLs.

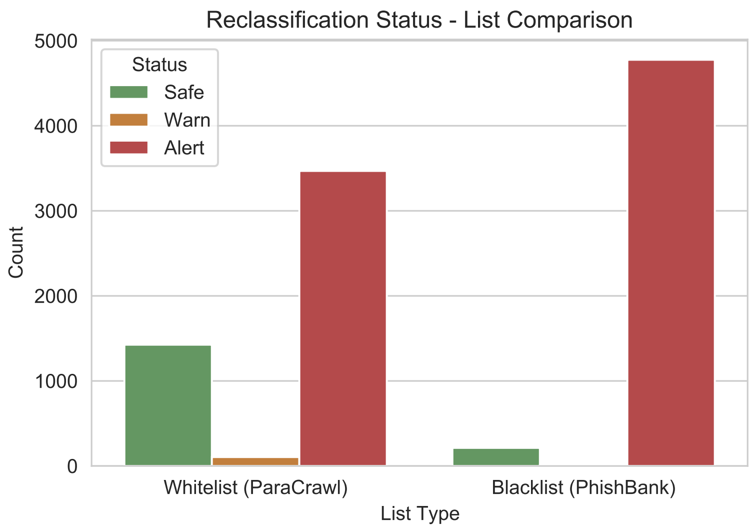


Figure C.2: The URL reclassification status following removal of the suspected broken heuristics as part of the Iteration with Real Data trial. These graphs illustrate limited change barring the addition of a small number of warn classifications.

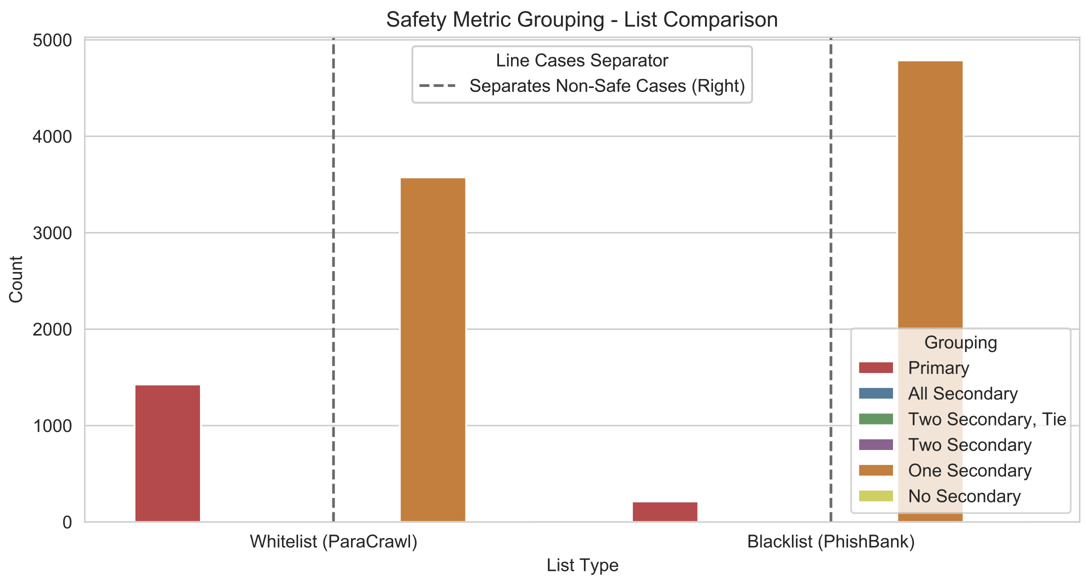


Figure C.3: Illustrated are the safe URL classifications for both the datasets used in the Iteration with Real Data trial. These classifications are grouped by the combinations of safety metrics which contributed to their classification (these cases are split by a line in the graph). Only the activation of the primary safety metrics contributed to URLs that were classified as safe. The graph also highlights the limited gains from adjusting the safety algorithm: since only one secondary/popularity safety metric regularly contributed to the safety classification, which occurred more frequently with blacklisted URLs. Therefore adjusting the secondary safety metrics would have decreased the false negative rate at the greater expense of the false positive rate.

Appendix D

URL Analysis Study - Additional Graphs

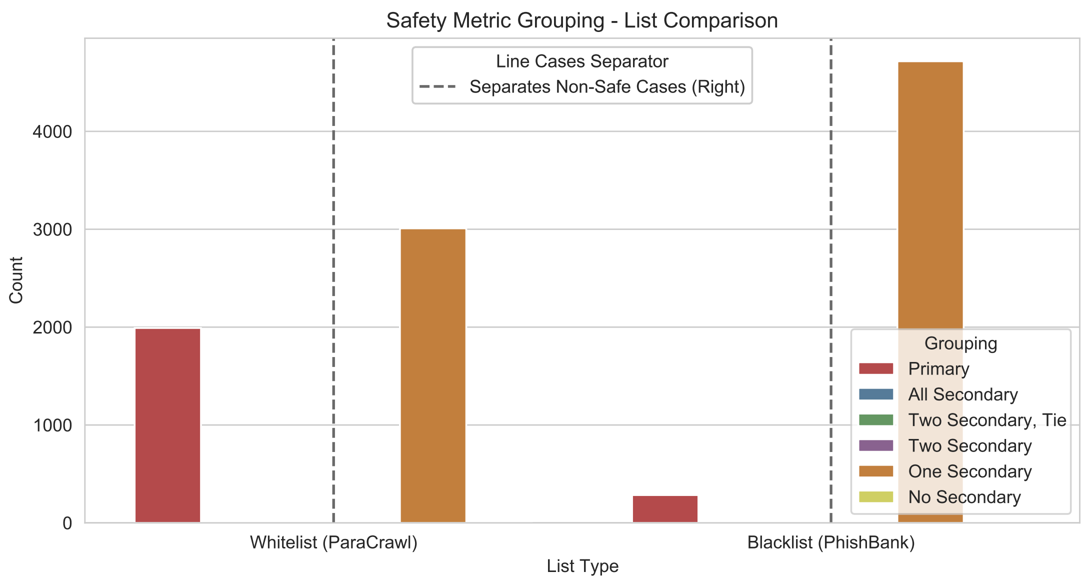


Figure D.1: The safe URL classifications for both Datasets A and B from the URL Analysis Study are illustrated. These classifications are grouped by the combinations of safety metrics which contributed to their classification (these cases are split by a line in the graph). This illustrates the limited but positive impact of adding the categorised data lists to the system when contrasted with Figure C.3.

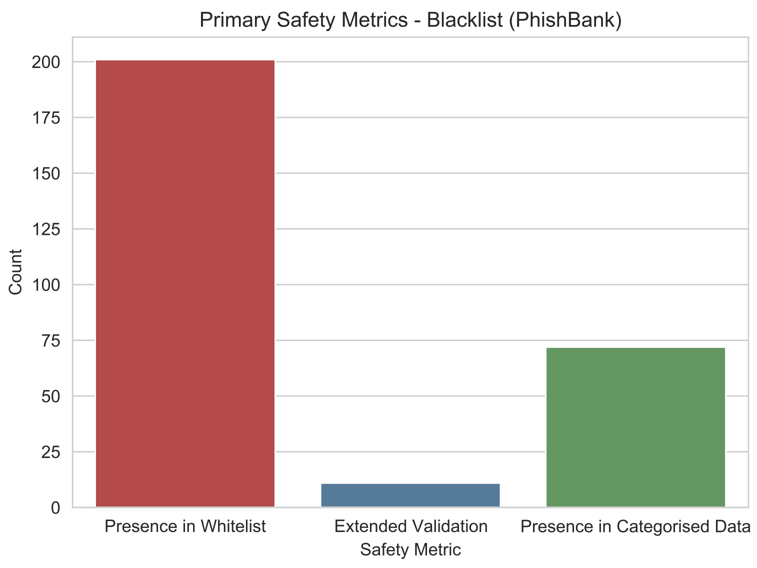


Figure D.2: A breakdown of the safety metrics for Datasets A and B, highlighting the small false negative rate arising from the URL Analysis Study. This also illustrates the amount of blacklisted URLs found to be present in whitelists and the impact of adding the categorised data lists, both factors which result in a higher false negative rate.