DESIGN AND DEVELOPMENT OF A MOBILE-BASED MALICIOUS URL DETECTION APPLICATION

An Information Technology Capstone Project

Presented to the Faculty of the College of Information and Computing

University of Southeastern Philippines

Bo. Obrero, Davao City

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY

Juarez, Leah C.

Ricafort, Sydney P.

Adviser

Tupas, Hermoso Jr. J.

June 2022

APPROVAL SHEET

The Information Technology Capstone Project entitled **DESIGN AND DEVELOPMENT OF A MOBILE-BASED MALICIOUS URL DETECTION APPLICATION** was prepared and submitted by **Sydney P. Ricafort** and **Leah C. Juarez,** in partial fulfillment of the requirement for the degree of Bachelor of Science in Information Technology major in Information Security, has been examined and is hereby recommended for an oral examination, acceptance, and approval.

Hermoso J. Tupas Jr.
Adviser

Approved by the Committee on Oral Defense

Ivy Kim D. Machica
Member

Cheryl R. Amante
Member

Accepted in partial fulfillment of the requirements for the degree of Bachelor of Science in Information Technology.

Ivy Kim D. Machica
Dean

ACKNOWLEDGEMENT

We, the researchers, would like to acknowledge and express our heartfelt gratitude and appreciation to the Almighty God and to the people who helped, supported, and encouraged us during and until we finished this capstone project.

First and foremost, we want to express our deepest gratitude to the Almighty God for the continuous guidance, good health, provision, blessings, wisdom, and strength, He has bestowed upon us. Because of Him, we persevered and overcame our dire circumstances during the making of this capstone project.

To the College of Information and Computing (CIC), the researchers want to thank the college for the opportunity to learn and present a capstone project. This opportunity pushed us to try our best and gain essential skills as soon-to-be IT practitioners.

To Sir Hermoso J. Tupas Jr., our dear adviser, we offer our warmest appreciation to him for the patience, encouragement, guidance, advice, and wisdom he has shared throughout our capstone journey. Because of him, our paper improved significantly, and our confidence increased that this capstone project would be finished on time.

To Ma'am Ivy Kim D. Machica and Ma'am Cheryl R. Amante, our panelists, their constructive feedback is highly appreciated. Their comments and recommendations push us not to settle for poor results.

To our classmates, the BSIT batch 2021-2022, and our friends, we want to let them know that their support and encouragement are really appreciated.

Finally, to our caring families, the Juarez and Ricafort family, especially to our parents, Fidel and Rosale Juarez; and Lorna and Nestor Ricafort, who have given us emotional and financial support not just during the making of this paper but throughout

our academic years, we, the researchers want to offer our profound gratitude to them and thank God for their existence in our life.

Our heartfelt gratitude belongs to all of them.

Glory be to God!

Leah & Sydney

ABSTRACT

Year over year, communication tools such as Social media are highly targeted by cybercriminals. Their schemes include the distribution of malicious URLs. A malicious URL is a URL that facilitates scams, frauds, and a cyberattack. By clicking on a malicious URL, a person can automatically download a malware program or a virus that has the ability to take over their devices or trick them into disclosing sensitive information on a fake website. End users who lack a fundamental understanding of information security are easier to be exploited by cybercriminals. One of the solutions for this kind of problem is using blocklist lookup; though effective, blocklists have a significant false-negative rate and lack the ability to detect newly generated malicious URLs. To address this problem, the researchers developed a mobile application called MalWhere, which can detect website links from image data. MalWhere also utilizes and combines the two known URL classification approaches; blocklisting and machine learning. With the blocklist service, MalWhere has access to a large number of Malicious URLs around the world. With machine learning, MalWhere can predict the classification of a URL, whether it is benign or malicious. The classification model was trained using 39 features and a supervised machine learning classifier called XGBoost. The classification ability of the model is mainly used to classify unknown and benign URLs to the blocklist service. Based on the conducted testing, MalWhere has an 88% accuracy rate in predicting the classification of a URL—whether it is benign or malicious.

Keywords: malicious URL, URL classification prediction, XGBoost, mobile application

TABLE OF CONTENTS

COVER PAGE
APPROVAL SHEET ii
ACKNOWLEDGEMENT iii
ABSTRACT v
TABLE OF CONTENTS vi
LIST OF FIGURES vii
LIST OF TABLES ix
LIST OF ABBREVIATION x
CHAPTER 1 INTRODUCTION 1
1.1 BACKGROUND OF THE STUDY 1
1.2 PROJECT CONTEXT 5
1.3 OBJECTIVES 7
1.4 SIGNIFICANCE OF THE STUDY 8
1.5 SCOPE AND LIMITATIONS 8
CHAPTER 2 REVIEW OF RELATED LITERATURE AND SYSTEMS 11
2.1 RELATED LITERATURE 11
2.2 RELATED SYSTEMS 31
CHAPTER 3 FRAMEWORK AND METHODOLOGY 37
3.1 CONCEPTUAL FRAMEWORK 37
3.2 METHODOLOGY 38
CHAPTER 4 RESULTS AND DISCUSSIONS 74
CHAPTER 5 SUMMARY, CONCLUSIONS AND RECOMMENDATIONS 85
REFERENCES 88
APPENDIX A 95
APPENDIX B 96
APPENDIX C 102
APPENDIX D 109
APPENDIX E 114
APPENDIX F 120
APPENDIX E 122

LIST OF FIGURES

Figure	Description	Page No.
1	Attack Scenario of Facebook Hacker	13
2	Structure of URL	14
3	Phishing Attacks Hosted in HTTPS every Quarter by PhishLabs	15
4	Extreme Boosting schematic representation	20
5	A comparison of the adjective ratings, acceptability scores, and school grading scales, in relation to the average SUS score	31
6	Conceptual Framework	37
7	RAD model	38
8	Project Timeline Snippet	40
9	MalWhere Palette	48
10	MalWhere Sign-in	49
11	Image Upload /Capture	49
12	Scan text URL	49
13	Navigation Drawer	49
14	Web Monitoring - Dashboard	50
15	Web Monitoring – Sign-in	50
16	Web Monitoring -View User Report Page	50
17	Web-based Monitoring -Map	50
18	MalWhere System Use Case	51
19	Mobile Application Flowchart	52
20	Web-based Monitoring System Flowchart	53
21	Sign-in	55
22	Main Screen	55
23	Splash Screen	55
24	Crop Image URL	55
25	Recognize text from image	55
26	Detect URL from text	55
27	Report URL screen	56
28	Navigation menu drawer	56
29	Report sent success pop-up message	56
30	Scan Results	56

31 About Screen....56
32 Sign-out/ Exit confirmation message....56
33 URL Classifying Screen....57
34 URL Classification pop-up message for malicious URL....57
35 URL Classification pop-up message for benign URL....57
36 Error Screen....57
37 Data Privacy Screen....57
38 Sign-in with Google SSO....57
39 MalWhere app source code snippet....58
40 Web Monitoring -Sign-in page....59
41 Web monitoring-dashboard....59
42 Web report monitoring page-picture view....60
43 Web report monitoring page....60
44 Web report monitoring page-classify user reported URL....61
45 Web report monitoring page-success message for manual classification.61
46 Web report monitoring page -download report....62
47 Web report monitoring page-downloaded report in excel file....62
48 Web monitoring map page....63
49 Web logout/sign- out....63
50 Web monitoring source code snippet....64
51 Show classification model features to remove code snippet....66
52 First classification model creation code snippet....68
53 Second classification model creation code snippet....69
54 Prediction result....70
55 Flask code snippet....71
56 API deployed in Heroku screenshot....72
57 Extract text from an image code snippet....74
58 Extract the long URL of a short URL code snippet and UI....75
59 Pop-up dialog of a URL classified as malicious and the blocklist service in combination with the classification model implementation code snippet. 77
60 Automated email response....78
61 Uploaded image sample....79
62 Captured image sample....80

LIST OF TABLES

Table	Description	Page No.
1	Examples of cyber attacks	11
2	Properties of different URL features	25
3	Confusion Matrix	28
4	Related Systems Comparison Matrix	35
5	User and System Requirements	41
6	Classification Model Feature List	46
7	Image upload test case	79
8	Capture image test case	80
9	Copy & paste test case	81
10	Non-Normalized Confusion Matrix	81
11	MalWhere Classification report summary	82

LIST OF ABBREVIATION

ML	Machine Learning
XGBoost	Extreme Gradient Boosted Tree
URL	Uniform Resource Locator
SUS	System Usability Scale
REST	Representational State Transfer
API	Application Programming Interface
IDE	Integrated Development Environment
TP	True Positive
FN	False Negative
TN	True Negative
FP	False Positive
OCR	Optical Character Recognition
SSO	Single sign-on

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF THE STUDY

As the utilization of the internet increases in our modern lives, the frequency of internet-related crimes *(e.g., cyber-attacks)* also increases. Cyberattacks have many forms; phishing, spamming, and malware attacks are few of the examples. As counted by a study, there are more than 2,200 cyberattacks a day, or about one cyberattack per 39 seconds [1]. With the truth about the cyberattack data, it only takes one click to possibly cause a data breach.

In the United States alone, it is said that there were 3,950 confirmed data breach incidents in 2020. According to International Business Machines (IBM), the leading causes of data breaches are malicious attacks which are the primary cause of data breaches (52%), followed by system glitches (25%), and lastly, human error (23%) [2]. The damages brought by these attacks are distressing for the victims and could be very expensive for organizations. The average cost of a data breach is around US$ 3.86 million. However, financial consequences differ significantly depending on the region, size of the organization, and industry [3].

Globally, cybercrime costs the world nearly US$600 billion, or 0.8 % of global gross domestic product (GDP) each year, as reported by the Center for Strategic and International Studies (CSIS) and McAfee in 2018 [4]. Unfortunately, the cybercrime costs are predicted by Cyber Security Ventures to increase up to US$10.5 trillion annually by 2025 [5].

Since 2020, because of the COVID-19 pandemic, people are online more than ever. Whether we work from home or attend classes online, we rely on technology and digital services [3] to operate remotely. However, for companies, remote working

increases cyber security risks. Without the security measures implemented in offices (e.g., anti-malware software) and being unknowledgeable of the basic cyber risks and security, people have become more vulnerable to cyber-attacks. According to an IBM security analysis in 2020, a remote workforce can increase the average cost of a data breach to almost $137,000, making the average total cost of $3.86 million go up to approximately $4 million [2]. In 2021, the average data breach costs increased to $4.24 million, which is the highest reported average cost in the entire history of the IBM and Ponemon Institute report [8].

An example case of a cyber-attack can be traced back to London. London Police revealed that £78 million had been lost due to 'clone firm' scams where criminals make a fake copy of actual investment firms; copying their name, address, telephone number, firm reference number, and even website and email addresses, with victims losing £452,421 on average, between January-December 2020 [9].

Nationally, here in the Philippines, cyber-attacks have already led to job losses in seven out of ten or 72% of organizations and can incur a potential economic loss of US$3.5 billion, or 1% of the country's total GDP of US$346.8 billion, as revealed by Frost & Sullivan Microsoft in 2018 [6], [7]. A study published by Cisco in 2021 revealed that 57 percent of small and medium-sized businesses (SMBs) in the Philippines experienced cyber-attacks in the previous 12 months, with each losing from over P25 million to over P50 million. Furthermore, they reported a 53 percent increase in cyber incidents during the pandemic, and 70% believe that a severe cyber incident could put their company out of business [10].

During this pandemic, one of the digital services that most internet users avail of is social networking services. Social networking services, commonly referred to as "social networking sites" or social media [11], are used by people as a medium through which they transmit, collaborate, and in some cases, live their lives [12], but for cybercriminals, it is an ocean to fish at. Year over year, Social media phishing attacks

are increasing, and they're getting more sophisticated every day [13]. In the first half of 2021, Vade declared Facebook as the first top favorite social media brand to impersonate by phishers and is the second top impersonated brand in all categories [14].

As the most availed services, social media (e.g., Instagram, WhatsApp, LinkedIn, Snapchat, Twitter, and Facebook) currently has many phishing links circulating on its platforms today [15]. As an example, Doevan listed 26 identified Facebook scams and malware attacks in his blog. These Facebook scams and malware versions are given a general term as Facebook viruses [16]. Most of these versions use malicious URLs to facilitate their attack, such as this version called “Facebook video virus”. The modus operandi of this attack is that users receive a message that contains “My Private video” or a similar message and a link to the alleged video.

Nevertheless, if clicked, users might immediately install malware. The malware will take control of the victim's account and automatically post malicious links with titles like "My private video," "My video," and "Private video" on the victim's timeline. Moreover, it tags random victim’s Facebook friends in these posts to draw their attention and invite them to click on the link. In addition, the virus also sends messages with a malicious link directly to the victim’s friends, thus making this threat spread rapidly and continuously unless not clicked [16].

However, links containing malware are not just seen on people’s friends' timelines or in their messages; they can also encounter infected links on Social Media Advertisements. In his blog, Vijayan cited a study by McGuire revealing that up to 40% of the malware infections on social media are from malicious ads, and 30% come from rogue apps and plug-ins. These distributed malware programs on social media platforms have infected one in five organizations. More than 12% of them have experienced data breaches because of the distributed malware on social media sites [17].

Another phishing technique used by cybercriminals is the abuse of short URLs or URL redirectors. Malicious short URLs, unlike long URLs, are difficult to detect using simple direct domain lookup or backlist checking. URL shorteners take long URL input and return an output of a short URL with the same landing page. This function is convenient for cybercriminals when they try to cover their malicious-looking URL and use the shortened URL in posting malicious links on Social Media, especially on Twitter [18].

The top 10 most abused redirectors as of September 22, 2021, according to SURBL.org, are the following: bit.ly; bit.do; ow.ly; goo.gl; x.co; rebrand.ly; tinyurl.com; t.co; is.gd; and ht.ly respectively [19] . According to Infosec Insights, data from an article indicates that 40.9% of phishing attacks used bit.ly and bitly.com URL shorteners [20].

Guan et al. believed that social media's friend-to-friend" application environment has proven to be a hazard to information safety and caused a surge of attacks from hackers. Since friends on social networking sites trust each other, these people will be less cautious of their friends' messages/emails/links, which are a potential danger. Social networking sites nowadays have become sources of malware, spam mail, and phishing activities, using the trust between people in the sites to lower a person's cautiousness towards potential internet attacks [21]. It has been claimed that Facebook spammers can earn $200 million from posting links [22], [23]. These malicious activities downgrade user experience and violate Facebook's terms of service. Furthermore, in rare cases, hoaxes have claimed human lives. Facebook has acknowledged that false news is harmful, and they are working to stop misinformation and false news on Facebook [22], [24].

Aside from the increasing attacks on social media platforms, other forms of communication are also highly targeted by cybercriminals. However, the bottom line is that malicious URL is used in various ways and is an integral part of a successful

cyberattack. It is used to steal or damage a person's or an organization's right or property, resulting in a considerable amount of financial loss and tarnished reputations. Thus, internet users need to be wary and vigilant about suspicious URLs.

One of the solutions for this kind of problem is using blocklist lookup, though effective blocklists have a significant false-negative rate [25] and cannot classify URLs unknown from its database. As an action to address this problem, we, the researchers, developed a mobile application called MalWhere. MalWhere utilizes and combines the two known URL classification approaches; blocklisting and machine learning. With the blocklist service, MalWhere has access to a large number of Malicious URLs around the world. With machine learning, MalWhere can predict the classification of a URL, whether it is benign or malicious. Our classification model was trained with 39 features and a supervised machine learning classifier called XGBoost. The classification ability of the classification model is mainly used to classify the URLs that are unknown and benign to the blocklist service. Unlike other detectors, users don't need to copy and paste the URL to the input field; instead, MalWhere gives users the option to input the URL by uploading the URL image, which helps avoid accidental clicks and eliminates the risks of automatic download and malware execution. The main aim of this app is to stop the process of the cyber-attack by hindering people from interacting with the malicious link.

1.2 PROJECT CONTEXT

During the first quarter of 2021, statistics show that financial institutions were the target of 24.9 percent of phishing attacks, followed by social media with 23.6 percent of attacks, and Webmail/SaaS with 19.6 percent of phishing attacks, making these three industries the most targeted in the world when it came to phishing [26]. Furthermore, according to Vijayan, a study by McGuire revealed that malicious ads account for up to 40% of malware infections on social media sites. In comparison, rogue apps and plug-

ins account for 30% [17]. This means that the attacks impersonate brands outside the messaging platforms and inside the messaging platforms legally.

Buber et al. cited five main reasons for people to fall for phishing [27], and these reasons are the following:

1. They are not aware of the URLs and their usage.
2. They do not know which URLs can be trusted.
3. They cannot see the target URL because the URL is redirected or hidden.
4. They accidentally click some URLs or do not have enough time to check the URL.
5. They are unable to distinguish between legitimate and phishing URLs.

To address this worldwide threat and help internet users not to fall victim to malicious URLs, the researchers developed MalWhere. MalWhere is a mobile application developed to assist internet users in classifying URLs. Furthermore, MalWhere is the researchers' response to the limitation of the traditional URL blocklist approach. The blocklist approach has a serious issue of high rates of false negatives and cannot detect newly generated malicious URLs [25]. To deal with its weakness, the researchers implemented another different approach to combine with it, and it is the machine learning approach.

With a machine learning approach, the researchers created a classification model which classifies and predict URLs, if it is malicious or benign. The classification model was trained using the 39 features that help the machine learning algorithm differentiate malicious URLs from benign URLs. These 39 features are composed of 37 lexical-based features and two host-based features. To ensure a high accuracy rate, the machine learning algorithm used in training the classification model is one of the most proven accurate classifiers and currently the most popular; the XGBoost [28]. The researchers designed the application to use the classification ability of the classification model to

the URLs that are labeled as unknown and benign by the blocklist service. Predicting the classification of URLs labeled benign to a blocklist service is an intervention to lessen false-negative results. The VirusTotal API is the selected blocklist service that was integrated into the mobile application.

On top of the powerful URL classification method, the researchers also implemented an intervention for accidental clicking of a malicious URL. This intervention is the use of Digital Image Processing and Text Recognition Technology which was integrated into the mobile application using Firebase ML Kit API. Instead of users copying and pasting the URL to the classifier's input field, users can have an option to take a picture or upload the image of the URL as another way to input a URL, avoiding accidental clicking on a malicious URL which may automatically download and run malware.

This mobile application will serve as an assistance for internet users in their efforts to be vigilant in avoiding clicking malicious URLs. This will be a new method for detecting malicious URLs on social media and other messaging platforms (e.g., SMS, Gmail, and Discord).

1.3 OBJECTIVES

The main objective of this study is to design and develop a mobile-based application that detects malicious URLs. Specifically, this study aims to:

1. Extract text from image data.
2. Extract the URL from the recognized text.
3. Automatically classify whether a URL is benign or malicious.
4. Monitor the reported URLs and extract the validated reported URLs for classification model update and improvement.
5. Test the accuracy of the mobile application.
6. Test the usability of the mobile application

1.4 SIGNIFICANCE OF THE STUDY

This study is beneficial for the following:

1. Researchers

This study gives empirical research results on the accuracy of the mobile application with regards to its used approach, which can be used as a reference for researchers in their related studies.

2. Mobile Phone Internet Users

This study will help internet users to have a malicious URL detector or URL classification tool to protect themselves against cyberattacks (e.g., phishing and malware) through a mobile application.

3. Students

This research may help students gain knowledge about malicious URLs and learn an approach on how to classify URLs.

1.5 SCOPE AND LIMITATIONS

The research is limited only to the following:

1. Develop and deploy a mobile application that classifies URLs, whether it is benign or malicious.
2. Develop and deploy a web-based monitoring system for managing reported URLs.
3. The mobile application classifies URLs, including shortened URLs.
4. The mobile application uses Unshort API, an API that extracts the full link of a shortened URL. However, the API does not support all URL shortener services. This API was developed by Siddiqui Affan and is published on GitHub.

5. The area of scope for the utilization of the mobile application is the internet or any platforms that offer messaging, or communication services and are only limited to URLs, not hyperlinks.
6. There are 3 URL input methods users can choose from when using the mobile application: image upload, camera capture, and copy & paste.
7. In terms of maliciousness, the researchers did not include the URLs of a webpage that has a piece of fake news because its contents are difficult to discern because only experts can determine if it is a piece of fake information.
8. For URL classification, the researchers used a combination of blocklisting and machine learning approaches. To be more specific, Virus Total was used as a blocklisting service (a service that gives access to a large database of labeled URLs through its API) in combination with a classification model that can predict the classification of a URL.
9. For the establishment of dataset reliability, the gathered labeled benign URLs are the only URLs which labels are rechecked before being used in training the classification model. This was done due to the limited time allotted for this project and VirusTotal Public API limitations. The use VirusTotal Public API limits requests to four per minute and 500 requests a day.
10. There is a total of 2 datasets created that were used on the intended 1 iteration classification model creation.
11. In creating the classification model XGBOOST (Extreme Gradient Boosted Tree), a supervised machine learning algorithm was used due to supporting works of literature that prove that it is suitable and accurate in classifying URLs, whether it is benign or malicious.
12. Thirty-nine URL features are used in creating the classification model. These 39 features are composed of 37 lexical-based features and 2 host-based features.
13. Python xgboost library with the scikit-learn library was used to implement XGBoost in creating the classification model. The classification model was trained using the default hyperparameters of XGBoost.

14. The final classification model that was used to combine with the blocklist service was the 2nd model, which was created using the 1st created model.
15. All APIs, libraries, and IDEs used in the project study are all open-source or free versions.
16. For the Text Recognition ability of the mobile application, Firebase ML Kit API was used.
17. For URL Detection, URL Detector API developed by LinkedIn Security Team was used.
18. The developed mobile application is only compatible with Android phones with versions 6.0 – 10.00.
19. For mobile application development, the minimum hardware requirements are 1.8GHz processors and at least 4GB of RAM. These requirements can run the IDE used by the researchers which is Android Studio. Java was used as the programming language in mobile application development.
20. For web application development, the minimum hardware requirements are 1.8GHz processors and at least 4GB of RAM. These requirements can run the IDE used by the researchers which is Visual Studio Code. HTML; CSS; PHP; JavaScript; MySQL; Bootstrap framework; jQuery; and Ajax are the programming languages and frameworks used in web application development.
21. For classification model development, the recommended hardware requirements are 8GB of RAM and an i5core processor at 2.59 GHz and up. These recommended specifications are capable of running the required computer software smoothly. The computer software used to develop the classification model is the Anaconda Prompt and Spyder. These two can be downloaded with other packages and libraries by downloading Anaconda Distribution.
22. Python 3 was used as the programming language in creating, training, testing, and saving the classification model.
23. The classification model was deployed in the form of a REST API. Flask, a python web framework, and Heroku were used for its creation and deployment.

CHAPTER 2
REVIEW OF RELATED LITERATURE AND SYSTEMS

The following literature, systems, data, and technologies included in this chapter, served as support and guiding principles to make this project possible.

2.1 RELATED LITERATURE

2.1.1 CYBER ATTACKS

Cybercriminals often use and target messaging platforms like social media to attain their malicious goals. An example of these cyberattacks is the distribution of various scams and malware programs through a malicious URL, promoted using Facebook [16], [29], [30]. Several examples of these cyberattacks are shown in Table 1:

Table 1. Examples of cyber attacks

Threat	Threat type	Distribution	Details
Facebook Video Virus [16]	Malware, scam/ Phishing	Spread through Messenger	Users receive a message with the phrases "My Private video" or something similar and a link to the allegedly released video. However, malware may be installed automatically if a user clicks on the link.
Congratulations! Your profile has been selected by Facebook [16]	adware	From installed freeware or plug-in.	Users are redirected to sketchy websites that display fake pop-up ads. The messages claim that Facebook has chosen users to receive a prize, but to receive it, they must provide financial information.

Facebook Change color scam [16]	Scam/ phishing	Messenger, spam emails	Users are offered to change their background color to a variety of colors. They must do so by clicking on a malicious link, directing users to a scam website.
WhatsApp trial service scam [29]	Scam/ phishing	Emails and SMS text messages from the WhatsApp app.	This trial service scam pretends to be a WhatsApp message and sends victims fake messages informing them that their one-year trial has ended and they must subscribe to extend it. Scammers advise their victims to access a "customer portal" and log in with their user credentials. After that, victims are asked to provide their banking details to purchase a monthly or annual subscription to the service. As a result, victims are charged with immense amounts of money for services that do not exist.
"The Nasty List"[30]	Scam/ Phishing	A direct message from the Instagram App	The user receives a message from a person that they are following. The message claims that the user has been featured on a "nasty list." The message contains a link formatted like this: @The_Nasty_List_randomnumber, which leads to a fake profile. This profile includes a description and a link where users can see this fake "Nasty list" and redirect to a fake Instagram login page. After credentials are entered, the hacker can now take over the victim's account and use it to promote the "Nasty List" scam.
UnionBank Php 10,000 loyal customer	Scam/ smishing	SMS text message	Users received an SMS text message with a malicious URL allegedly from UnionBank offering

Valentine's day treat [31]			Php 10,000 as a Valentine's treat for being a "loyal customer" of the bank. Upon clicking the link, the victim is redirected to a phishing site that imitates the UnionBank login page. The attackers redirect victims two to three times. In addition, if observed, these URLs used for redirection have different domains.

These attacks listed in Table 1 are just examples of cyberattacks happening on messaging platforms. The list shows that cybercriminals do not just use the brand names of reputable apps and companies but also the platforms itself. In figure 1, a general Facebook attack scenario is shown.

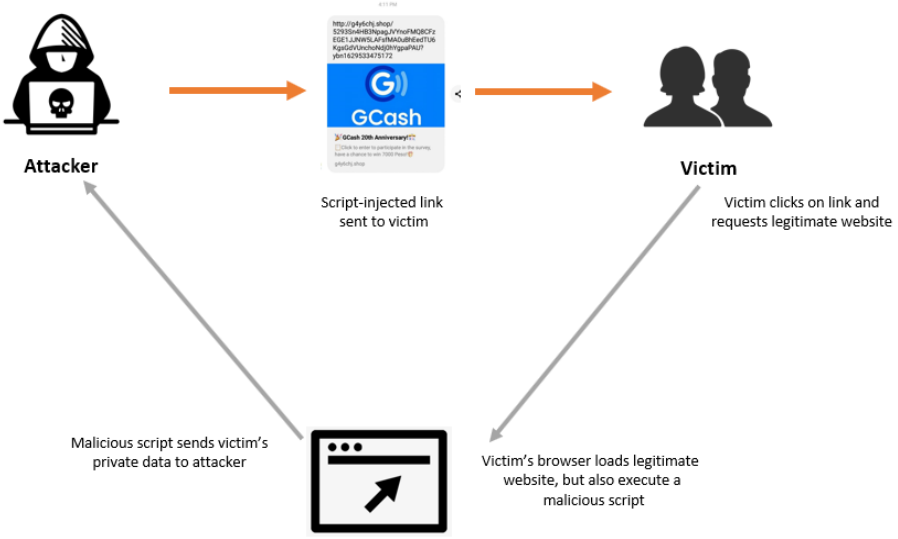


Figure 1. Attack Scenario of Facebook Hacker

As shown in figure 1, most of these cyberattack techniques are facilitated and start with a malicious URL attached to a post or message by an attacker. Once clicked, it redirects the user to a spoof page. After the victim has submitted their credentials through the spoof page, the credentials are then sent to the

attacker; in other cases, clicked malicious URLs do not redirect victims to spoof pages; instead, it automatically downloads malware to the victim's device to obtain the victim's credentials and use it to take over the victim's devices or accounts.

2.1.2 URL

A URL is a web address, a series of letters and numbers that leads to a website. Figure 2 shows the structure of a URL. The first portion of a URL is the protocol (i.e, "http" or "https") that is used to reach the page. The letters after the final period of a hostname are the Top-Level Domain (TLD). It is also called as domain suffix (e.g., ".com " or ".ph"). These parts of the URL can tell people what the website is for or who created it. It can also help people decide whether or not a website is trustworthy enough to utilize in research [32].

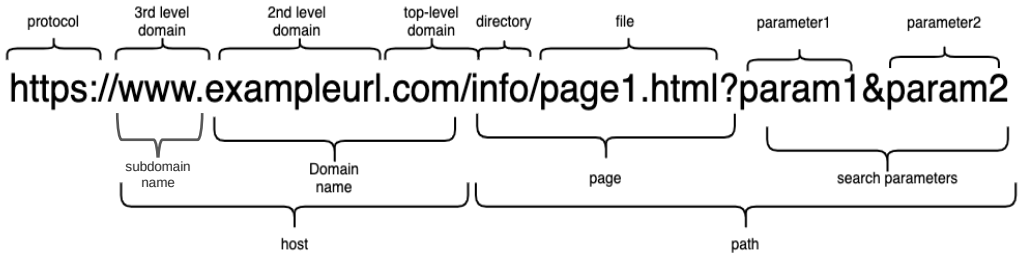


Figure 2. Structure of URL

The web page's host server is identified by the fully qualified domain name (FQDN). FQDN is made up of a registered domain name (second-level domain) and a suffix (top-level domain). A hostname is made up of two parts: a subdomain name and a domain name. The subdomain portions are entirely under the control of the phisher, who can set any value they want. The URL may also include path and file components, which the phisher can change at any time. The phisher has complete control over the subdomain name and path. Buber et al. termed these URL parts as "Free URL". Any "Domain Name" that has not been registered before can be registered by the attacker. This portion of the URL can only be changed once. The free URL can be changed at any time by the

phisher to create a new URL. The "Domain Name" is a unique part of the website, which is why security defenders have a hard time detecting phishing "Domain Names." When a domain is identified as fraudulent, it is simple to prevent a user from accessing it [27]. According to the Anti-Phishing Working Group (APWG), the TLDs that had a unique second-level domain used for phishing were “.COM”, “.UK”, “.ORG”, “.NET”, “.XYZ”, “.LIVE”, “.BR”, “.LINK”, “INFO” and “.ME”, respectively [33].

PhishLabs, an APWG contributor, has been tracking and studying the use of HTTPS on phishing sites. HTTPS encrypts data sent between a user's browser and the website they visit, ensuring that communications are secure. HTTPS is essential, especially for websites that handle password-protected accounts. Investigating HTTP on phishing sites discloses how phishers deceive Internet users by exploiting an Internet security feature to make them look legit. Phishing sites that use HTTPS are increasing every quarter, and the last quarter of the year 2020 has been recorded with the highest increase so far, as shown in figure 3 [33].

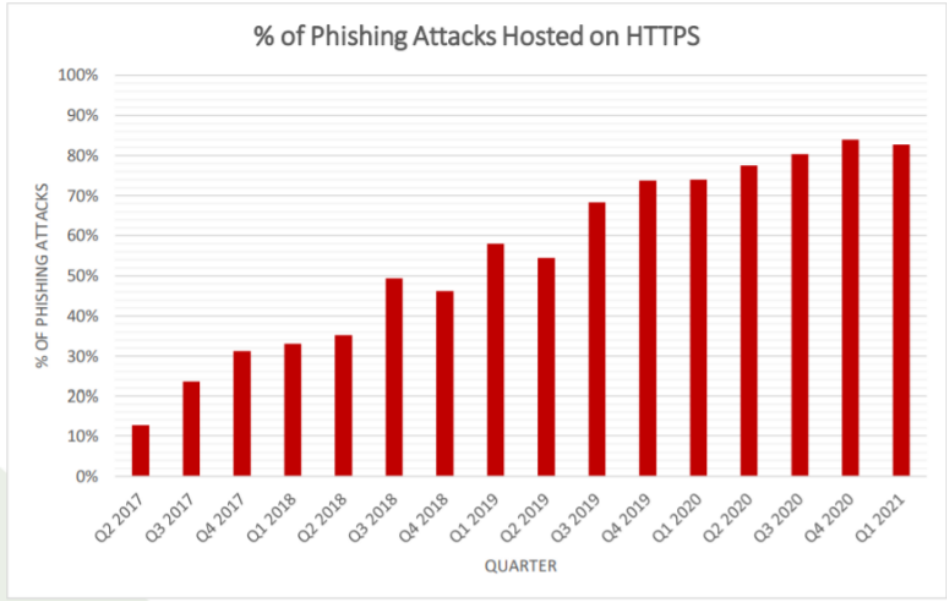


Figure 3. Phishing Attacks Hosted in HTTPS every Quarter by PhishLabs

2.1.3 TYPES OF MALICIOUS URL

These cyberattacks on messaging platforms (e.g., SMS, Gmail, and Messenger) often used malicious URLs. A malicious URL facilitates scams, frauds, and a cyberattack. By clicking on an infected or malicious URL, people can download malware such as a Trojan or virus that can take control of their devices or trick them into disclosing sensitive information on a fake website [34]. These Malicious URLs can be the following:

a.) spam [18], [22] - irrelevant messages sent to large number of people online;

b.) scam [18], [22]- an online fraud to mislead people;

c.) phishing [18], [35]- is online fraud to obtain credentials;

d.) malware [18], [35]- auto downloadable content designed to damage the system; or

e.) defacement [35]- modified websites or webpages in which its content was maliciously added, removed, or changed.

2.1.4 MACHINE LEARNING

Many researchers have studied about malicious URL detection for a long time. Several comparative studies are already conducted to analyze the drawbacks and determine the most proven and accurate URL detection approach among the existing approaches. An example of these comparative studies is the comparative study published in 2019 by Seena Thomas and Lekshmi A R, which determined the different methods or approaches to detecting malicious URLs. The approaches are Blocklist Approach, Heuristic or Rule-based Approach, and Machine Learning Approach. According to the authors, among the three approaches, machine learning techniques are better and more suitable for detecting malicious URLs [36].

Machine learning is the study of teaching a computer program or algorithm how to improve over time at a given task. In terms of research, machine learning can be viewed through theoretical and mathematical modeling, but in other view, it is the study of how to create applications that demonstrate iterative improvement. There is a variety of ways to conceptualize this idea, but the three most accepted categories are supervised learning, unsupervised learning, and reinforcement learning [37].

In Supervised Learning, labeled datasets are used to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model is well fitted, which happens during the cross-validation process [38] (e.g., face recognition).

Unsupervised learning is the opposite of supervised learning. It does not use labels in training. Instead, the algorithm would be fed with a large amount of data and given the tools necessary to understand the properties of the data. It can then learn to group, cluster, or organize the data so that it can be understood or interpreted by a human or other intelligent algorithm [37] (e.g., video recommendation).

Reinforcement learning is not a type of unsupervised learning, even though it does not use labels in training. In this type of learning, the algorithm's behavior is shaped through a sequence of rewards and penalties that are based on whether the algorithm’s decisions toward a defined goal are right or wrong. Unlike supervised learning, where the algorithm needs to model behavior based on examples, in reinforcement learning, the algorithm is allowed to behave freely, allowing it to discover which actions maximize reward and minimize penalty through trial and error [39] (e.g., self-driving cars).

Among the three types of Machine Learning, reinforcement learning is the only one that other researchers do not use to classify malicious URLs because it is not suited for classification problems.

2.1.5 MACHINE LEARNING ALGORITHM USED FOR MALICIOUS URL DETECTION/ CLASSIFICATION

In the present times, many machine learning algorithms are already introduced and used by many researchers in training their models to achieve high prediction accuracy systems in classifying benign and malicious URLs. Logistic Regression, Stochastic Gradient Descent, Naïve Bayes, K-Nearest Neighbors, Decision Tree, Random Forest, Support Vector Machine, Multilayer perceptron, C5.0, C4.5, and Gradient Boosting are just examples.

In terms of classification accuracy, many researchers have already compared these algorithms, and the majority of them showed that Random Forest attains the highest prediction accuracy and performs better than other machine learning algorithm classifiers [18], [40]–[44]. However, Kandolkar & Usgaonkar published a study in 2021. Their study is about finding the best-performing model by training various machine learning classifier models on the dataset created to predict malicious websites. The machine learning classifiers they have selected to compare are Logistic Regression, Support Vector Machine, K-Nearest Neighbor, Naïve Bayes, Decision Trees, Random Forest, and XGBoost. The study measured and compared the performance level of these machine learning classifier models. Their study concluded that out of the seven different classifiers that were evaluated, the XGBoost classifier model gave the best accuracy of 85.60%, followed by the Random Forest Classifier model with an 85.40 % accuracy rate [35].

Another study conducted by Binji in 2021 talks about finding the best ensemble algorithm for the classification of Malicious URLs. The author

compared seven machine learning classifiers, namely: Logistic Regression, Naïve Bayes, Decision Tree Classifier, Random Forest Classifier, Ada Boost Classifier, LightGBM (Light Gradient Boosting Machine) Classifier, and XGBoost Classifier. After the experiment, it was revealed that LightGBM Classifier and XGBoost Classifier have the best performance and were at par. However, XGBoost Classifier shows the highest accuracy level out of all models, with an accuracy level of 99.81 and a training time of 4.38 seconds, compared to LightGBM, with a 99.80 accuracy level and a training time of 0.72 seconds [45]. Based on this result, LightGBM is apart from being a highly accurate classifier, it is fast or time-saving than XGBOOST; however, the researchers still used XGBoost due to comments of some authors that LightGBM has less documentation available, that is why it's not popular [46].

Other authors, Chen and Guestrin, mentioned XGBoost in the competition scenario; according to them, a machine learning competition hosted by Kaggle in 2015 had many winning participants that used XGBoost; in fact, XGBoost was used in 17 solutions out of 29 winning solutions in the competition. In addition, last KDD Cup 2015 competition, XGBoost was also used by every winning team in the top 10 [47].

Because it has been proven to be an appropriate algorithm for URL classification and of its high accuracy reputation, the researchers have chosen to use XGBoost as the machine learning algorithm to train the classification model.

2.1.6 XGBOOST CLASSIFIER

XGBoost stands for Extreme Gradient Boosting. It is a machine learning algorithm and a decision tree-based ensemble learning method that uses the gradient boosting framework. XGBoost enhances the base Gradient Boosting framework through system optimization and algorithmic enhancements.

The algorithm was developed by Tianqi Chen and Carlos Guestrin as a research project at the University of Washington. Their research paper entitled "XGBoost: A Scalable Tree Boosting System " was presented at SIGKDD Conference in 2016. Since its release, this algorithm has not only won numerous Kaggle competitions but has also served as the driving force behind several cutting-edge industry applications. As a result, a strong community of data scientists contributing to the XGBoost open-source projects was formed. The algorithm can be utilized to solve regression, classification, ranking, and user-defined prediction problems [28]. Malicious URL classification is an example of a classification problem that can be solved by XGBoost.

XGBoost use ensemble tree method that uses the gradient descent architecture to boost weak learners. An ensemble method refers to the method where trees/model are built consecutively and individual trees/model are summed sequentially or combined rather than being all trained in isolation from one another to perform the final one. To avoid the error of the previous tree/model the next tree/model tries to lower the loss (difference between actual and predicted values) from the previous tree/model [48] [49]. Figure 4 shows the sequential schematic representation of XGBoost [50].

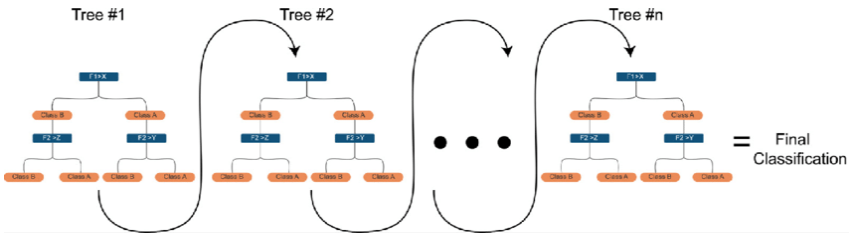


Figure 4. Extreme Boosting schematic representation

XGBoost was implemented by Tianqi Chen and Carlos Guestrin as an open-source library which is designed to be efficient, portable, and easy to implement. XGBoost repository is on GitHub and has more than 500

contributors as of June 2022, working together on improving the XGBoost open-source packages for different programming languages. XGBoost supports multiple programming languages such as Python, R, Java, Scala, and C++ [51]. In this project study, XGBoost was implemented using the XGBoost python library with Python 3 as the programming language to create, train and test the classification model.

2.1.7 FEATURES USED FOR MALICIOUS URL DETECTION

In malicious URL detection, there are two classes; “Malicious” and “Benign”. The quality of the training data, which is dependent on the quality of feature representation, is critical to the success of a machine learning model. The feature representation process can be further divided into two steps [52]:

(1) Feature Collection: This is an engineering-oriented phase that collects relevant URL information. This includes information such as whether the URLs are block listed, features extracted from the URL String, information about the host, website content such as HTML and JavaScript, and popularity data.

(2) Feature Preprocessing: In this phase, the URLs’ unstructured information (e.g., textual description) is appropriately formatted and converted to a numerical vector, which can then be fed into machine learning algorithms. The numerical data, for example, can be used as-is, and Bag-of-words is frequently used to represent textual or lexical content.

In building the detection mechanism, the system calculates the selected features according to the needs and purposes with labels and uses them for training. [27]. Researchers have proposed several types of features that can be used to provide useful information when detecting malicious URLs. In the paper “Malicious URL Detection using Machine Learning: A Survey” , the researchers grouped these features into Blocklist Features, URL-based Lexical Features,

Host-based features, Content-based Features, and Others (Context and Popularity). [52]. The feature groups that the researchers gathered from their survey are the ff:

1. Blocklist Features [52]

It is noted that blocklisting as a decision-maker in detecting malicious URLs is prone to serious high false negatives. However, a researcher cited that it can also be a feature. The blocklist feature alone is not good as other features, but when joined with other features, the overall performance of the prediction model was improved [52].

2. Lexical Features [35], [52]

Lexical features are features derived from the URL name's properties (or the URL string). The motivation is that a URL's malicious nature should be able to be identified based on how it "looks". In practice, these features are combined with others, such as host-based features, to boost model performance [35], [52].

Example [42] :

- length of the URL
- length of hostname
- number of characters consisting of the hostname
- length of path, (e) domain token count
- path token count
- average domain token length,
- average path token length
- count of Security sensitive words
- number of dots
- presence of IP address, and

- presence of .exe in URL

3. Host-based features [35], [52]

These features are extracted from the URLs' host-name properties. They provide details about the malicious hosts' properties, location, identity, and management style. [52].

Example [42] :

- safe browsing
- rank of host
- country
- site popularity

4. Content-based Features [27], [52]

These features are those obtained upon downloading the entire webpage. These are "heavy-weight" features when compared to URL-based features because many data must be extracted, and security concerns may arise. However, it is natural to assume that having more information about a specific webpage will result in a better prediction model [52].

5. Other Features [52]

A. Context Features

These features include content-related features such as lexical and statistical properties of the tweet; the context of the tweet features such as time, relevance, and user mentions; and social features such as following, followers, location, tweets, retweets, and favorite count [52].

B. Popularity Features.

Some other features used were designed as heuristics to measure the popularity of the URL. One of the earliest approaches uses statistical techniques to detect malicious URLs. Popularity features include page-based features such as page rank, domain-based features such as presence in the white domain table, and type-based features such as obfuscation types, which are all examples of these [52].

Sahoo et al. stated that there is a wide range of information that can be obtained from a URL. Crawling and converting the unstructured data to a machine learning compatible feature vector is a process that needs a lot of time and resources. While adding more information to models with the right regularization, gathering a large number of features is often impractical. Several host-based features, for example, could take several seconds to obtain, making their use in a real-world setting impractical. Another example he mentioned is the Kolmogorov Complexity, which necessarily requires comparing a URL to multiple databases of malicious and benign URLs, which may be impractical with billions of URLs. He also added that Blocklist, Context, and Popularity features require additional dependencies and thus have a higher collection overhead; meanwhile, the content features have the highest security risk due to potential malware that may be explicitly downloaded while trying to obtain these features. Meanwhile, the collection of the lexical features does not suffer from that issue and is very efficient, as they are direct derivatives of the URL string[52].

To help others wisely decide what features they should use when training a model, they presented the properties of every feature by category [52], which is shown in Table 2.

Table 2. Properties of different URL features

Features	Category	Criteria					
		Collection Difficulty	Risk	External Dependency	Collection Time	Processing Time	Feature Size
Blacklist	Blacklist	Moderate	Low	Yes	Moderate	Low	Low
Lexical	Traditional	Easy	Low	No	Low	Low	Very High
	Advanced	Easy	Low	No	Low	High	Low
Host	Unstructured	Easy	Low	No	High	Low	Very High
	Structured	Easy	Low	No	High	Low	Low
Content	HTML	Easy	High	No	Depends	Low	High
	JavaScript	Easy	High	No	Depends	Low	Moderate
	Visual	Easy	High	No	Depends	High	High
	Other	Easy	High	No	Depends	Low	Low
Others	Context	Difficult	Low	Yes	High	Low	Low
	Popularity	Difficult	Low	Yes	High	Low	Low

2.1.8 URL COLLECTION

Part of creating a classification model is to collect URLs for training and testing. There are already many publicly available datasets that are used by other researchers in collecting benign and malicious URLs (e.g., spam URLs, malware URLs, and phishing URLs) in their studies. For the researchers to choose reliable data, several research studies were gathered that tackles Malicious URL detection/classification. Here are the few datasets which the other researchers have chosen to collect their URLs and services for their blocklist lookup approach to classify benign and malicious URLs.

- VirusTotal [18], [22], [53], [54]

  VirusTotal is a free service that analyzes files and URLs for viruses, phishing, and other kinds of malicious content. VirusTotal inspects items by aggregating the output of about 70 antivirus scanners and URL/domain block listing services. Examples of these services are PhishTank, ESET, Avira, Google Safe Browsing, Cyren, and Kaspersky. People can avail the services or submit and retrieve scan results to VirusTotal by using the VirusTotal webpage, browser extension, and API [55].

- Google Safe browsing [18], [21], [22], [56]

  Google Safe Browsing is one of the trusted blocklisting services by some researchers to check URLs or web resources if it is malicious [56].
- SURBL [18], [21], [22]

  SURBL contains a list of spam, phishing, malware, and other malicious websites that appear in unsolicited messages[57].
- Phish Tank [18], [22], [35], [58]–[61]

  PhishTank is a service website dedicated to sharing phishing URLs. Suspicious URLs can be sent to PhishTank for verification. The data in PhishTank is updated hourly [58].
- OpenPhish [40], [54], [59]

  OpenPhish is a repository of active and new phishing URLs [62].
- Zone-H [54]

  A dataset for defacement URLs.
- Datasets from Kaggle platform [40], [44], [58]

  Researchers also used labeled datasets from the Kaggle platform to collect malicious and benign URLs. Kaggle is a community of data scientists and machine learning practitioners where they compete, discuss, and share their code and data/datasets from their research or study. Examples of these datasets used by researchers are the Malicious and Benign Websites [63] and the Malicious_n_Non-Malicious URL [64].
- Web of Trust (WOT) [21], [22]

  WOT system computes and shows existing websites' reputations. WOT API returns a reputation score for a given domain. The reputation computation used user ratings and third-party sources like malware, phishing, scam, and spam blocklists.WOT states that to keep ratings

more reliable, the system tracks each user's rating behavior before deciding how much it trusts the user [22], [65].

- University of New Brunswick URL Dataset (ISCX-URL2016) [35] – This is a dataset from a study entitled "Detecting Malicious URLs Using Lexical Analysis", which has a collection of spam, phishing, malware, and defacement URLs [66].
- Dmoz.org [59]–[61]
  Dmoz is a collection of human-edited directories of web contents [67].
- URLhaus [58] - URLhaus is a project from abuse.ch that aims to share malicious URLs used for malware distribution. URLhaus dataset lets people download and submit malicious URLs from its database [58], [68]

Some of the URLs stored in some of these listed datasets are crowdsourced, so other researchers established the reliability of their dataset by using multiple URL blocklist services like Google Safe Browsing and/or VirusTotal when labeling the collected malicious URL as a malicious URL. Other researchers also conduct a manual inspection for crowdsourced malicious URLs that are not detected as malicious by other blocklist services [53].

For the collection of datasets and labeled URLs that were used in the project study, three datasets were selected and utilized, which were all from the Kaggle platform. These 3 datasets are named as Malicious_n_Non-Malicious URL dataset[64], malicious URL dataset[69], and SMS Collection Dataset[70]. The different types of URLs that were collected are benign, phishing, malware, spam, and defacement URLs. The researchers kept the 15,000 labeled benign URLs and 15,000 malicious URLs from the gathered URLs. To establish reliability, manual inspection and Virus Total API were used to check if any

benign URLs are marked as malicious by other blocklist services queried by VirusTotal. The formed dataset was used to train the classification model.

2.1.9 CONFUSION MATRIX

When measuring the performance of a classification model, a confusion matrix can be used to show the summary of the number of correct and incorrect predictions made by the classification model. Table 3 depicts the confusion matrix.

Table 3. Confusion Matrix

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

True Positive (TP), is an instance denoting where the predicted and actual true values are identical (i.e., positive). False Positive (FP) is a case where the predicted value was incorrect. Although the actual value is negative, the classification model predicted a positive value. True Negative (TN) is a case where the predicted value matches the actual value (i.e., negative). False negative (FN) is a case when the expected value is incorrectly predicted. The classification model predicted a negative result, while the actual value was positive.

By utilizing the results displayed in the confusion matrix, the performance of a classification model can be measured by calculating the performance metrics. This study used accuracy, precision, recall, and F1-score as the performance metrics [71]. The following is their description and formula.

• Precision is the ratio of true positives to the sum of a true positive and false positive [35]. Precision = TP/(TP + FP)

• Recall is the ratio of correct true positive classifier decisions to the all true positive [35]. Recall = TP/(TP + FN)

• F-measure (F1) represents the previous metrics precision and recall combined[35]. F1 = 2 ∗ (precision ∗ recall) / (precision + recall)

•Accuracy is the measure of the true predictions divided by the total [35]. Accuracy = (TP + TN)/(TP +TN + FP+ FN)

• Support is the number of actual occurrences of the class in the specified dataset [35].

• Macro average is the simple mean of all class scores [72]. Macro Avg = (class score 1 + class score 2+…class score N)/number of different classes .

• Weighted average is the mean of the sum of all class scores after multiplying their respective class proportions. Weighted Avg = (class score 1*total of class 1+ class score 2* total of class 2 + …class score N * total of class N)/total of all class values.[72].

2.1.10 USABILITY

When developing a system or digital product, its usability should be tested, asking the users' perspective. In assessing the usability of the system, the researchers utilized the System Usability Scale. The System Usability Scale (SUS), developed by John Brooke in 1986, is a simple and quick means to assess the usability of systems and applications. SUS is made up of ten statements, each of which has a five-point scale ranging from Strongly Disagree to Strongly Agree.

SUS is perceived as a practical and reliable method for assessing usability, and it can be applied to a wide range of digital products and services to assist UX practitioners in determining whether a design solution has an overall flaw. SUS is primarily used to provide an overall usability evaluation measurement, which also reflects the specified standard of evaluation measurement that should cover specified by ISO 9241-11, which are the following: (1) effectiveness – will users be able to attain their goals? ; (2) efficiency refers to how much work and resources are put into reaching such goals; and (3) satisfaction- was your experience satisfactory? [73], [74]. The ten statements are composed of five good and five bad alternate statements. The following are the 10 SUS statements.

Q1. I think that I would like to use this system frequently.

Q2. I found the system unnecessarily complex.

Q3. I thought the system was easy to use.

Q4. I think that I would need the support of a technical person to be able to use this system.

Q5. I found the various functions in this system were well integrated.

Q6. I thought there was too much inconsistency in this system.

Q7. I would imagine that most people would learn to use this system very quickly.

Q8 found the system very cumbersome to use.

Q9. I felt very confident using the system

Q10. I needed to learn a lot of things before I could get going with this system

The SUS score ranges from 0 to 100. However, the meaning of each score must be understood intuitively, lacking a single SUS score in an absolute sense. Hence, Bangor et al. conducted research to address this problem and added an adjective rating scale to the System Usability Scale. Using the adjective rating scale with the SUS, individual SUS scores can now be interpreted with less difficulty [75]. Figure 5 shows the SUS score compared by the authors with their adjective rating scale and other score rating scale, which are school grade scales and acceptability ranges.

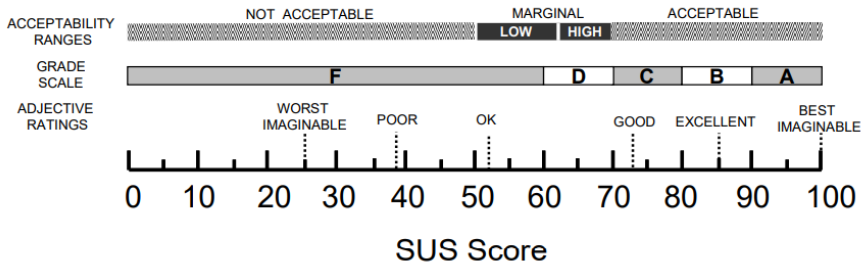


Figure 5. A comparison of the adjective ratings, acceptability scores, and school grading scales, in relation to the average SUS score

2.2 RELATED SYSTEMS

2.2.1 TEXT RECOGNITION TECHNOLOGY

Optical character recognition (OCR), also known as text recognition, is a software technology that electronically recognizes text in an image file or physical document, such as a scanned document, and converts it into a machine-readable text form to be used for data processing [76]. Some popular software that integrates OCR are Mathway, Google image search, and Google translator. In regards to the mobile application, the researchers recognized and decided to make OCR a security measure because of its ability to extract text from images. The integration of OCR on the developed application will enable users to check if a URL is dangerous through their camera or uploaded images instead of trying to copy and paste the unknown URL, which can lead to an accidental click that can result in an automatic download and execution of malware. To the best of

the researchers' knowledge, in regards to fetching URLs, MalWhere is the first to integrate OCR in a mobile application for URL classification.

2.2.2 MALICIOUS URL DETECTORS

- Facebook Inspector (FbI) - a REST API-based browser plug-in to detect malicious Facebook posts in real-time. These malicious posts that they are identifying are two kinds of malicious posts, (a) posts containing malicious URLs and (b) posts that violate Facebook's terms or community standards (containing hate speech, profanity, etc.) or are spam concerning the event under consideration. To identify malicious posts in real-time, FbI uses class probabilities obtained from two independent supervised learning models based on a Random Forest classifier. These supervised learning models, which use only publicly available features, are based on a feature set of 44 features and achieve an accuracy of over 80%. Facebook Inspector processed 0.97 million posts in the first nine months of its public launch, with an average response time of 2.6 seconds per post and over 2500 downloads [22].
- Malicious URL Detection on Facebook - the author came up with a system that can effectively detect malicious URLs and spam messages. In gathering data from selected top twenty fan pages on Facebook, they analyzed 49110 Facebook wall posts. They came up with seven criteria to distinguish different kinds of internet threats. When analyzing various internet threats, the four criteria used to distinguish internet threats yielded an accuracy of 94.94%, providing a good system for distinguishing malicious URLs. When they integrated this system with the Bayes Classification Model, the TPR reached 95.07%, while the FNR was only 3.56%. When including a short URL sample, the system's detection rate lowered. Nevertheless, the method used in the system is still proven effective in detecting malicious URLs [21].
- A Lexical Approach for Classifying Malicious URLs – is a classification system based on pure lexical analysis of URLs (uses J48 model classifier

and n-gram approach). It classifies malicious web pages' URLs with 99.1% accuracy and a 0.4% false-positive rate, an F1-Score of 98.7, with an average response time of 0.62 milliseconds. However, currently, their system rejects shortened URLs, and in data gathering, they only collected two kinds of malicious URLs: phishing and malware. The researchers envision this system to be integrated with an existing real-time security system such as a firewall, browser extension, or as a pre-filter to a comprehensive detection scheme [59].

- bit.ly/malicious: Deep Dive into Short URL-based e-Crime Detection – Gupta et al., in this paper, proposed a detection mechanism for malicious Bitly links. They identified specific Bitly features and grouped them into Non-Click based features and Clicked based features and put them together with some domain-specific features to classify a Bitly URL as malicious or benign. Each Bitly URL took about 50 seconds to compute these features. Other short URL services also provide similar user and link metrics as used in their feature set; therefore, the proposed mechanism is scalable to other services. On a mixed dataset of clicked and non-clicked links, the classifier (random forest classifier) predicted malicious links with the best accuracy of 80.40 percent. The classifier improved its accuracy by 83.51 percent on the mixed dataset and 86.41 percent on the Non-Click dataset by removing Click-based features. This clearly shows that their algorithm is capable of detecting malicious Bitly links not only after they have been clicked but also before they have been clicked. This shows that in addition to the blocklists and other spam detection filters, Bitly-specific feature sets can also be used to detect malicious content [18].
- MALICIOUS UNIFORM RESOURCE LOCATOR DETECTION – The inventors describe a technique to use training data to train classification models to detect malicious Uniform Resource Locators (URLs) that target authentic resources (e.g., Web page, Web site, or other network locations

accessed via a URL). The techniques train the classification models using one or more machine learning algorithms (e.g., SVM). The malicious URL determination may be based on one or more lexical features (e.g., brand name edits distances for a domain and path of the URL) and/or site/page features (e.g., domain age and a domain confidence level) extracted. In regards to short/redirected URLs, the system has a Redirection Module that redirects the training URLs to their actual URLs so that feature extraction can be performed on the actual URLs [77].

- URLVOID Website Reputation Checker – is a website that analyzes a website using 30+ blocklist engines and online website reputation services to help identify fraudulent or malicious websites and websites implicated in malware and phishing incidents [78].
- Detecting Mobile Malicious Webpages in Real Time – Amrutkar et al. created a browser extension that provides real-time feedback to users using kAYO. kAYO, the new static analysis technique (which uses static features of mobile webpages derived from their HTML and JavaScript content, URL, and advanced mobile-specific capabilities.), was designed and developed by the authors to detect mobile malicious webpages which provide 90% accuracy in classification. kAYO was modeled using logistic regression, which is a popular machine learning binary classifier [56].

2.2.3 COMPARISON MATRIX

Table 4 shows the comparison matrix of related systems, comparing the approaches and features of the existing related detectors to the developed mobile application.

Table 4. Related Systems Comparison Matrix

	Deployed in:	Use OCR	Use blocklist and ML approaches to classify URL	Use ML approach only to classify URL	Use ensemble method	Real-time	Has intervention for malicious short URL problem	Accuracy rate
DESIGN AND DEVELOPMENT OF A MOBILE-BASED MALICIOUS URL DETECTION APPLICATION	Mobile application	✓	✓	✗	✓	✗	✓	88%
Facebook Inspector (FbI)	Facebook REST API-based browser plug-in	✗	✗	✓	✓	✓	✓	80.56 %
A Lexical Approach for Classifying Malicious URLs	Not yet deployed (Approach)	✗	✗	✓	✗	Not yet deployed	✗	99.01 %
bit.ly/malicious: Deep Dive into Short URL based e-Crime Detection	Not yet deployed (Technique)	✗	✗	✓	✗	Not yet deployed	✓	86.41 %
MALICIOUS UNIFORM RESOURCE LOCATOR DETECTION	Technique to be patented	✗	✓	✗	✓	✗	✓	N/A
URLVOID Website Reputation Checker	Web reputation and Blocklisting Website & API	✗	✗	✗	N/A	✗	✗	N/A
Detecting Mobile Malicious Webpages in Real Time	Firefox mobile browser extension	✗	✓	✗	✗	✓	✓	90%

As observed on the comparison matrix, many studies did not publicly deploy their systems yet, or if they have, it is customized to detect malicious URLs in certain platforms only (e.g., Facebook platform) [21], [22] or can be

used in certain compatible browsers (e.g., Mozilla Firefox) or not created for free public use. In addition, some of the existing detection tools require users to input URLs in a text field [78], which may cause the automatic download of malware when accidentally clicked during the copy and pasting process. Furthermore, not all systems created interventions or emphasizes how to specifically solved the difficulty of classifying shortened URLs, interventions such as extracting the original URL of a shortened URL [77] or including specific features to collect distinct information that classifies a malicious shortened URL from other benign shortened URL [18].

CHAPTER 3
METHODOLOGY

This chapter discusses the conceptual framework and methodology that is used to create the project study. The conceptual framework is shown in figure 6.

3.1 CONCEPTUAL FRAMEWORK

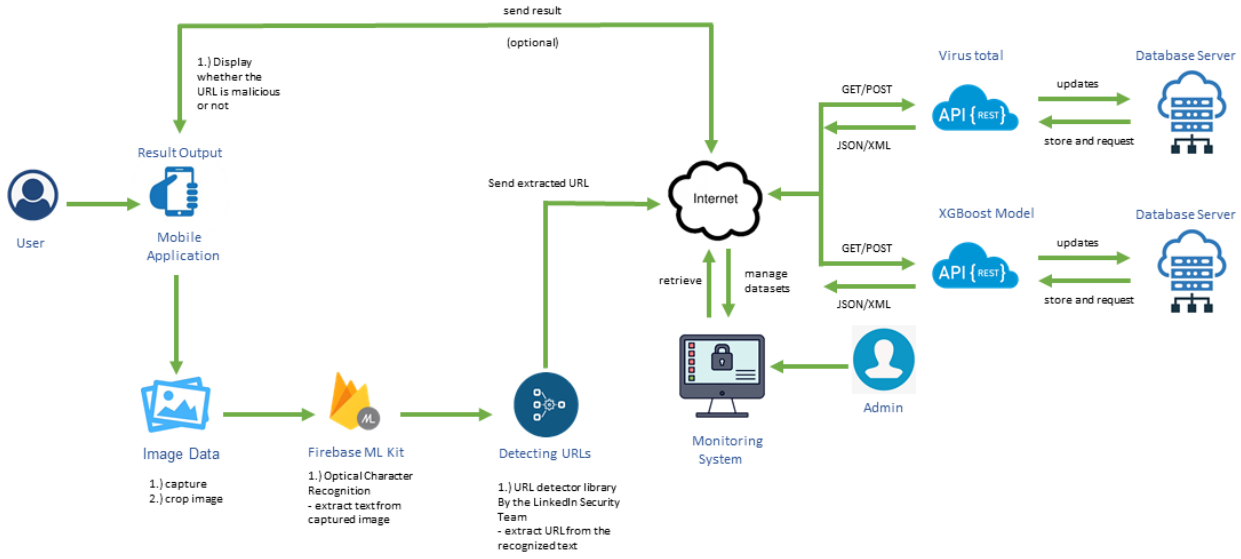


Figure 6. Conceptual Framework

Figure 6 shows the conceptual framework of the study. The structure covers the components needed for detecting malicious URLs from an image. First, through the mobile application, a user can provide an image containing the URL using the built-in camera or image uploader. Second, the image will be then converted into a block of text using Firebase On-device text recognition API. The application will then extract the URL from the converted block of text using an open-source Java Library URL detector from LinkedIn Security Team. Third, the extracted text URL will then be sent to the blocklist service/s for classification. If the classification result is unknown or benign the URL will be sent to the classification model and then sends a response which will be displayed through the mobile application, whether the URL is malicious or benign.

Lastly, when the user reports a URL, the mobile application will send the created report with all its detail to the web monitoring system. The web monitoring system application then displays the reported URLs with its information to be accessed and managed by the administrators. Later after the administrator validated the true classification of a URL an automatic email response will be sent to the reporter and the list of validated URLs will be used as a new dataset to update the classification model.

3.2 METHODOLOGY

3.1.1 SDLC

Figure 7 shows the Rapid Application Development (RAD) Model, which is a development methodology followed by the project study. Rapid application development (RAD) is a popular agile software development project management strategy [79]. It emphasizes rapid prototyping and feedback over long and complex development and testing cycles. Rather than needing to start a development schedule from the start each time, developers can make multiple iterations and update a software program quickly with rapid application development. RAD model arose from developers' realization that the old waterfall model was ineffective. The primary issue in the waterfall model is that once the program is in the testing phase, changing the software's core functions and features becomes difficult. As a result, software owners may left with software that may or may not meet their evolving requirements [80]. RAD model had been chosen because the researchers believed that the earlier the system

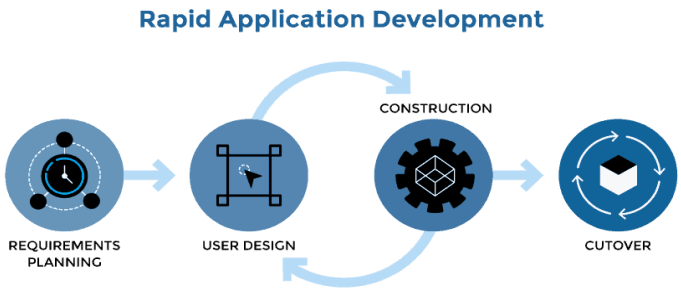


Figure 7. RAD model

integration, the earlier the researchers could identify any errors and complications within integrations and force immediate resolutions.

RAD Model could be divided into four phases which are Requirements Planning, an iterative process of User Design phase and Rapid Construction & Feedback phase, and Cutover.

Phase 1: Requirements Planning

Unlike the waterfall model, in the RAD model, a detailed list of requirements from end-users is not necessary to gather from the start; rather, a broad requirement or high priority requirements are asked from end-users. This broad nature gave the researchers the time to gather specific requirements at different stages of the development cycle.

In this study, the activities that were performed are:

A. Determining project constraints;
B. Creation of project timeline; and
C. Gathering of main requirements and requirements revision/update

The following sections explain the activities mentioned above in more detail.

A. Determining project constraints

The following are the determined project constraints:

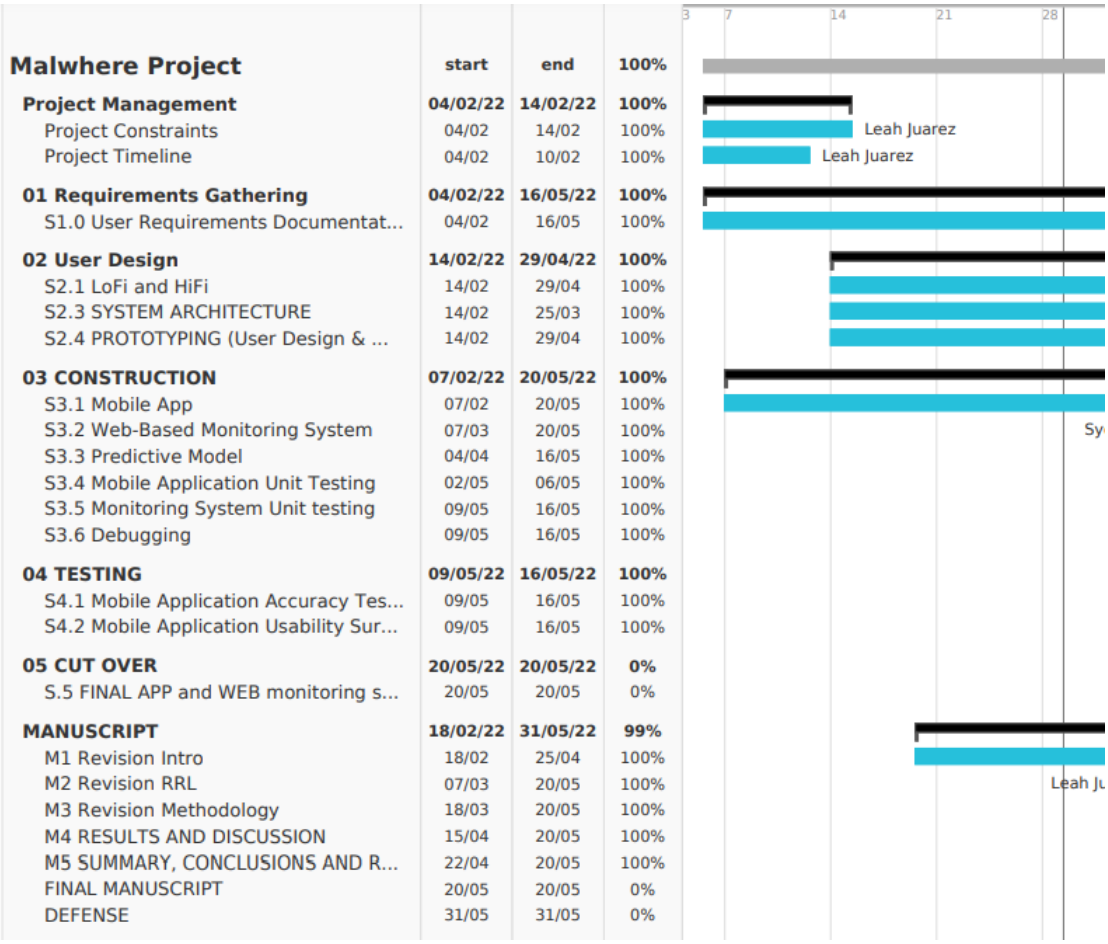
- Scope
    - Development of a mobile application
    - Development of a web-based monitoring system
    - Development of a classification model
    - Manuscript
    - Total number of members: 2 people
- Time

- February 2022-May 2022 (4 months)

- Budget
  - Php 1,376

B. Project Timeline

Figure 8 shows the project timeline. The whole picture of the project Timeline can be seen in Appendix A.



Malwhere Project	start	end	100%
Project Management	**04/02/22**	**14/02/22**	**100%**
Project Constraints	04/02	14/02	100%
Project Timeline	04/02	10/02	100%
01 Requirements Gathering	**04/02/22**	**16/05/22**	**100%**
S1.0 User Requirements Documentat...	04/02	16/05	100%
02 User Design	**14/02/22**	**29/04/22**	**100%**
S2.1 LoFi and HiFi	14/02	29/04	100%
S2.3 SYSTEM ARCHITECTURE	14/02	25/03	100%
S2.4 PROTOTYPING (User Design & ...	14/02	29/04	100%
03 CONSTRUCTION	**07/02/22**	**20/05/22**	**100%**
S3.1 Mobile App	07/02	20/05	100%
S3.2 Web-Based Monitoring System	07/03	20/05	100%
S3.3 Predictive Model	04/04	16/05	100%
S3.4 Mobile Application Unit Testing	02/05	06/05	100%
S3.5 Monitoring System Unit testing	09/05	16/05	100%
S3.6 Debugging	09/05	16/05	100%
04 TESTING	**09/05/22**	**16/05/22**	**100%**
S4.1 Mobile Application Accuracy Tes...	09/05	16/05	100%
S4.2 Mobile Application Usability Sur...	09/05	16/05	100%
05 CUT OVER	**20/05/22**	**20/05/22**	**0%**
S.5 FINAL APP and WEB monitoring s...	20/05	20/05	0%
MANUSCRIPT	**18/02/22**	**31/05/22**	**99%**
M1 Revision Intro	18/02	25/04	100%
M2 Revision RRL	07/03	20/05	100%
M3 Revision Methodology	18/03	20/05	100%
M4 RESULTS AND DISCUSSION	15/04	20/05	100%
M5 SUMMARY, CONCLUSIONS AND R...	22/04	20/05	100%
FINAL MANUSCRIPT	20/05	20/05	0%
DEFENSE	31/05	31/05	0%

Figure 8. Project Timeline Snippet

D. Gathering of main requirements and requirements revision/update

The main requirements gathering was collected through research and analysis. To be specific, the gathered user and system requirements were based on the objectives of the project study and research about users' reasons why they fall victim to malicious URLs. The user and system requirements are shown in Table 5.

Table 5. User and System Requirements

User Requirements	System requirements	
Mobile Application		
1. User can sign in to the mobile application	Functional	Non-Functional
	FR1-Login flow the mobile application shall allow users to sign in using their Google account or sign in anonymously.	**NFR1- URL classification combination sequence.** The mobile app shall classify the final URL input using an integrated blocklist and machine learning classifiers. The combination shall operate in sequence. The primary URL classifier must be the blocklist service, given that it has a large dataset of malicious URLs. The secondary classifier must be the classification model. The model will be used when the URL blocklist returns a benign or unknown result. This technique is an attempt to lessen the false-negative rate of a blocklist.
2. Can input URL through the camera, screenshot, and copy and paste.	**FR2- Input image** the mobile app shall accept images in different formats such as jpeg, jpg, and png.	**NFR2- Security** - the mobile app must be implemented with google SSO to allow users to authenticate their accounts.
3. Prevents accidental clicks	**FR3- crop image**	**NFR3- Usability** - The mobile application shall provide ease of use

	the system shall allow users to crop an image in different aspect ratios.	
4. Automatically classify if a URL is benign or malicious with accuracy.	**FR4- Extract text from the image** the mobile application shall extract text from the image.	
5. Let users report malicious URL	**FR5- Extract URL from a text** The mobile application shall be able to extract URLs from recognized characters in a standard format. URLs that are formatted in HTML 5 scheme, IPv4 address, IPv4 Decimal, IPv6 address, and IPv4-mapped are included.	
6. Easy to use	**FR6- Extract the long URL of a shortened URL** the mobile application shall detect a short URL and extract its long URL.	
9. View status result description	**FR7- Copy & paste URL input method** the mobile application shall have an input text field as another option for URL input.	
8. Users can sign out from the mobile application	**FR8- URL classification** the mobile application shall classify the extracted final URL, whether benign or malicious, using the integrated and combined blocklist and machine learning classifiers.	
	FR9- User URL Report	

	The mobile application shall allow a user to submit a report and require them to fill in required information before enabling them to submit a report.	
	FR10- Scan result The mobile application shall allow the user to see the classification result from the blocklist service and the classification model(display both results if applicable).	
	FR11- Scan meaning The mobile application shall allow a user to see the meaning of the classification result.	
	FR12- Sign out The mobile application shall allow a user to sign out from the session.	
Web-based Monitoring System		
1. Admin can sign in to the monitoring system	**SFR1- View Dashboard** The system will enable the admin to view the summarized information of the reported URL through graphical representation.	**SNFR1- Access security** The system will implement authentication using username and password to restrict database and information access from unauthorized persons.
2. Admin can view and manage reported URLs.	**SFR2-Manage reported URL** The system will enable the admin to view, confirm the URL classification, download and send automatic email responses regarding the actual	**SNFR2- Availability** The system shall be available any time of the day or throughout "normal operating times" to be able to perform its features and function.

	classification of the reported URL	
3. Admin can sign out from the monitoring system	**SFR3- View Map** The system will enable the admin to view reporters' locations using geotagging technology. To get a view of the expansion of the or a malicious URL	**SNFR3- Usability** The system shall provide ease of use
	FR4- Sign out The system will enable a user to revoke their active sessions.	**SNFR4- Confidentiality** The system shall protect all data and information and permit only authorized users to access the data by disabling the directory listing.

The user requirements were updated based on the results gathered from users' feedback which were collected through testing during phase 3 (Construction).

The deliverables in this phase are user and system requirements and the project Timeline.

Phase 2: User Design

In this phase, the goal was to design the mobile application and web application based on general software UI/UX and template designs in consideration of the user and system requirements. In addition, in this phase, the researchers also designed the system architecture and selected existing technologies and publicly available datasets to be used in mobile development. Some activities of this phase are simultaneously done with the next phase, the construction phase. In this study, the activities that were performed are:

A. Text recognition API selection;

B. Blocklist service/s for blocklist approach selection;
C. Dataset for classification model training selection;
D. Feature for classification model training selection;
E. Machine learning algorithm selection;
F. UI/UX design; and
G. System architecture design

The following sections explain the activities mentioned above in more detail.

A. Text recognition API selection

For the mobile application to have a text recognition ability, the researchers used and integrated the Firebase ML Kit API free version. The ML Kit has a general-purpose API suitable for recognizing text in images, such as text on a street sign [81]. Due to time constraints, the researchers did not conduct text recognition API products comparison, to find the best text recognition API. The researchers proceeded to choose this API because it is free and has accuracy.

B. Blocklist service/s for blocklist approach selection

The chosen blocklist service for the blocklist approach is Virus Total. Virus Total is frequently used by other researchers to get the labels of their unlabeled URLs [18], [22], [53], [54]. It queries about 70 antivirus scanners and URL/domain block listing services and uses JSON for requests and responses, including errors [55].

The researchers selected this API because it has a free version, is used by many researchers, and has the power of multiple blocklist services combined.

C. Dataset for classification model training selection

From manual collection and the three datasets collected from the Kaggle platform, hundreds of thousands of URLs were collected; however, 15,000 benign URLs and 15,000 malicious URLs (i.e.,

phishing, malware, defacement, spam) were only kept. These three datasets that were selected are named as Malicious_n_Non-Malicious URL dataset [64], malicious URL dataset[69], and SMS Collection Dataset [70]. These three datasets contain gathered URLs from different URL datasets (e.g., malicious URL dataset [69] contains URLs from ISCX-URL-2016 dataset, Malware domain blocklist dataset, Faizan git repo, Phishtank, and PhishStorm datasets.) which were published in the Kaggle platform for others' utilization for their machine learning studies and experiments.

With an equal number of benign and malicious URLs, the researchers obtained a balanced data set.

D. Feature for classification model training selection

Table 6 shows the final features that were used to extract information to train the classification model.

Table 6. Classification Model Feature List

Feature		Datatype	Return value
Lexical-based			
1	url_length	Int	URL length
2	use_of_ip	Int	IP address presence: 1-yes,0-no
3	path_length	Int	Path length
4	path_to_urllength_ratio	float	path_to_urllength_ratio
5	count_dir	Int	No. of directory
6	fd_length	Int	First Directory length
7	count_embed_domian	Int	No. of embedded URL
8	short_url	Int	Short URL presence: 1-yes,0-no
9	count-lowercase	Int	No. of lowercase alphabet in URL
10	lower_to_urllength_ratio	float	lower_to_urllength_ratio
11	count_uppercase	Int	No. of uppercase alphabet in URL
12	upper_to_urllength_ratio	float	upper_to_urllength_ratio

13	count-digits	Int	No. of numbers in the URL
14	count-letters		No. of letters in the URL
15	digit_to_urllength_ratio	float	digit_to_urllength_ratio
16	letters_to_urllength_ratio	float	letters_to_urllength_ratio
17	count-specchar	Int	No. of special characters in the URL
18	specchar_to_urllength_ratio	float	Special characters : URL
19	count-www	Int	No. of www in URL
20	count.	Int	No. of dots in URL
21	count@	Int	No. of @ in URL
22	count%	Int	No. of % in URL
23	count?	Int	No. of ? in URL
24	count-	Int	No. of hyphen in URL
25	count=	Int	No. of equals in URL
26	count#	Int	No. of # in URL
27	count;	Int	No. of semicolon in URL
28	count_	Int	No. of dots in URL
29	http_or_https	Int	2-http;0=https;1=no protocol result
30	entropy	float	Measurement of the randomness or disorderliness of the URL
31	tld_length	Int	Top-level domain length
32	tld_count	Int	No of Top-level domain
33	host_length	Int	Hostname length
34	host_count-	Int	No. of hyphen in Hostname
35	host_count_	Int	No. of underscore in hostname
36	sub_domain_count	Int	No. of sub domain
37	sus_url	Int	Suspicious Words: 1-yes; 0-no
Host-based			
38	days_since_reg	Int	Number of days since the site has registered
39	days_since_exp	Int	Number of days since the site has expired

In total, there are 39 features which 37 are lexical-based, and the other 2 are host-based. This activity was simultaneously performed with

the construction phase under the “creation of classification model” activity.

E. Machine learning algorithm selection

Based on the conducted literature review, Extreme Gradient Boosted Tree (XGBOOST) is a robust, accurate, and easy-to-implement supervised machine learning classifier. This classifier is most used by winning participants in machine learning competitions and has better performance in comparison with other algorithms. XGBoost is a type of supervised machine learning algorithm and is an optimized implementation of Gradient Boosting, which was first introduced by Chen and Guestrin. [35], [47]. It avoids overfitting/bias by optimizing the standard Gradient Boosting algorithm through parallel processing, tree pruning, missing value handling, and regularization [28]. This classifier was used to train and test the classification model using the same set of 39 features listed and described in Table 6.

F. UI/UX Design

Figure 9 shows the palette which was adopted from IDYLUM labs, a website that checks website vulnerabilities for security testing.

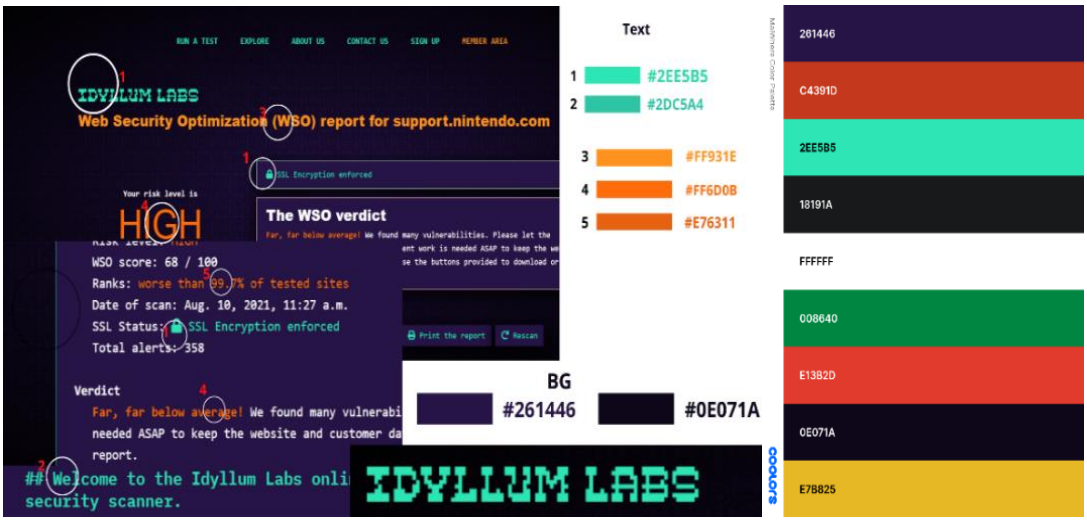


Figure 9. MalWhere Palette

- Mobile Application UI/UX Design

Figure 10-13 is the UI/UX main design for the mobile application.

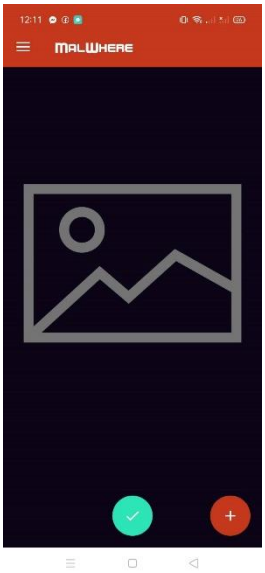


Figure 11. Image Upload /Capture

Figure 10. MalWhere Sign-in

Figure 12. Scan text URL

Figure 13. Navigation Drawer

- Web-Based Monitoring System UI/UX Design

Figure 14-17 is the UI/UX main design of the web monitoring system, which is a modified template from bootstrap.

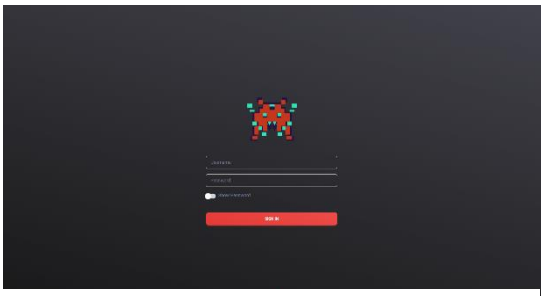


Figure 15. Web Monitoring – Sign-in

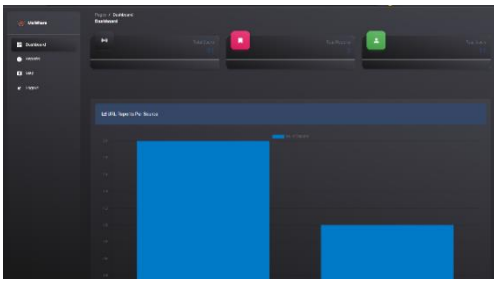


Figure 14. Web Monitoring - Dashboard

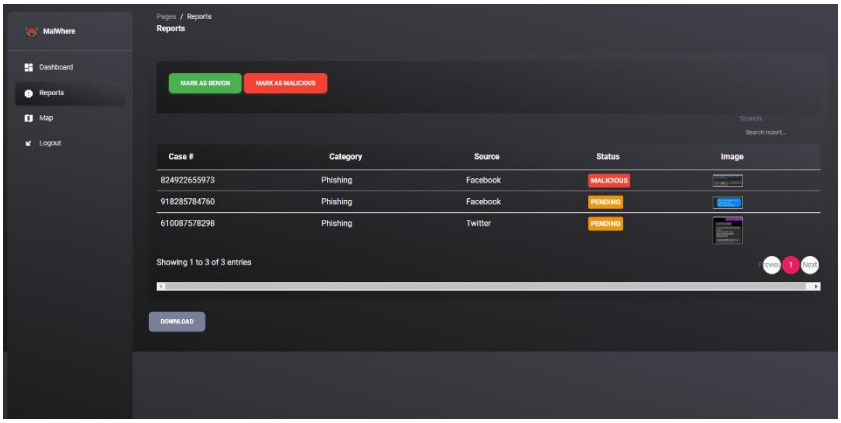


Figure 16. Web Monitoring -View User Report Page

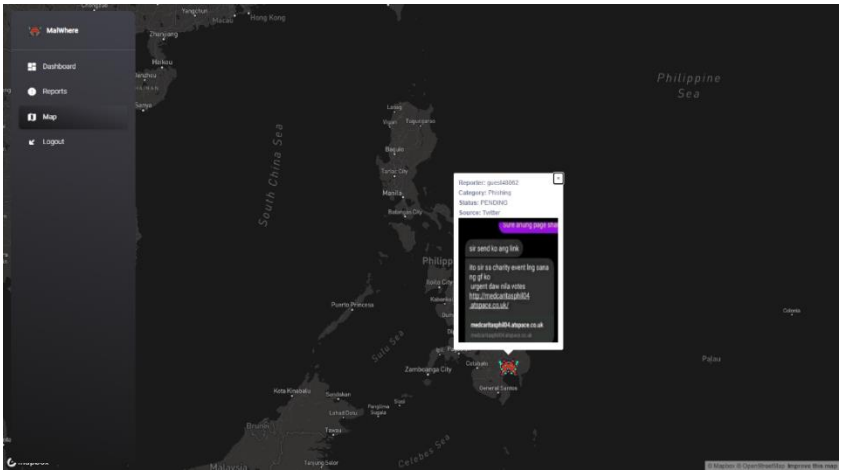


Figure 17. Web-based Monitoring -Map

G. System Architecture

- Use Case

Figure 18 shows the use case of the whole system. In the system, there are two types of users; the “user” who uses the mobile application and the “admin” who is the only user of the web-based monitoring system.

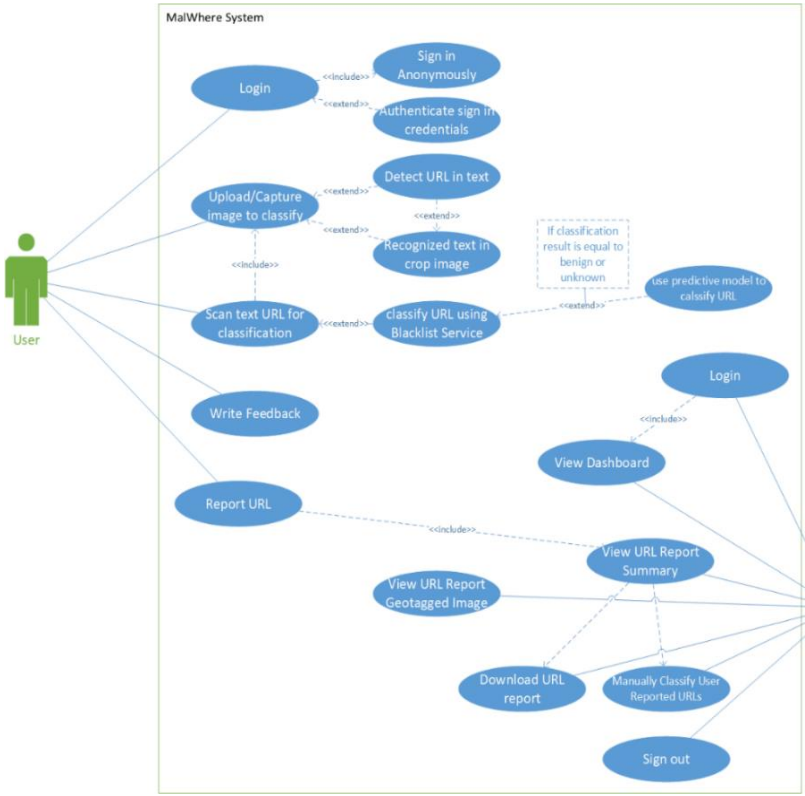


Figure 18. MalWhere System Use Case

Users can log in anonymously or use their Google account for identity verification. As observed in the figure 18, there are shapes and lines. The oval shapes contain the course of action and supporting functions that reflect the general case when using the MalWhere system. On the other hand, the lines represent interactions. The solid line represents the direct interaction of the user to an action. Meanwhile, the broken line is used to connect function to function relationships. Independent function extends help to dependent function. A dependent function is included because its

function cannot process on its own without the help of the independent function/s.

- Mobile application Flowchart

The mobile application's flowchart is shown in Figure 19.

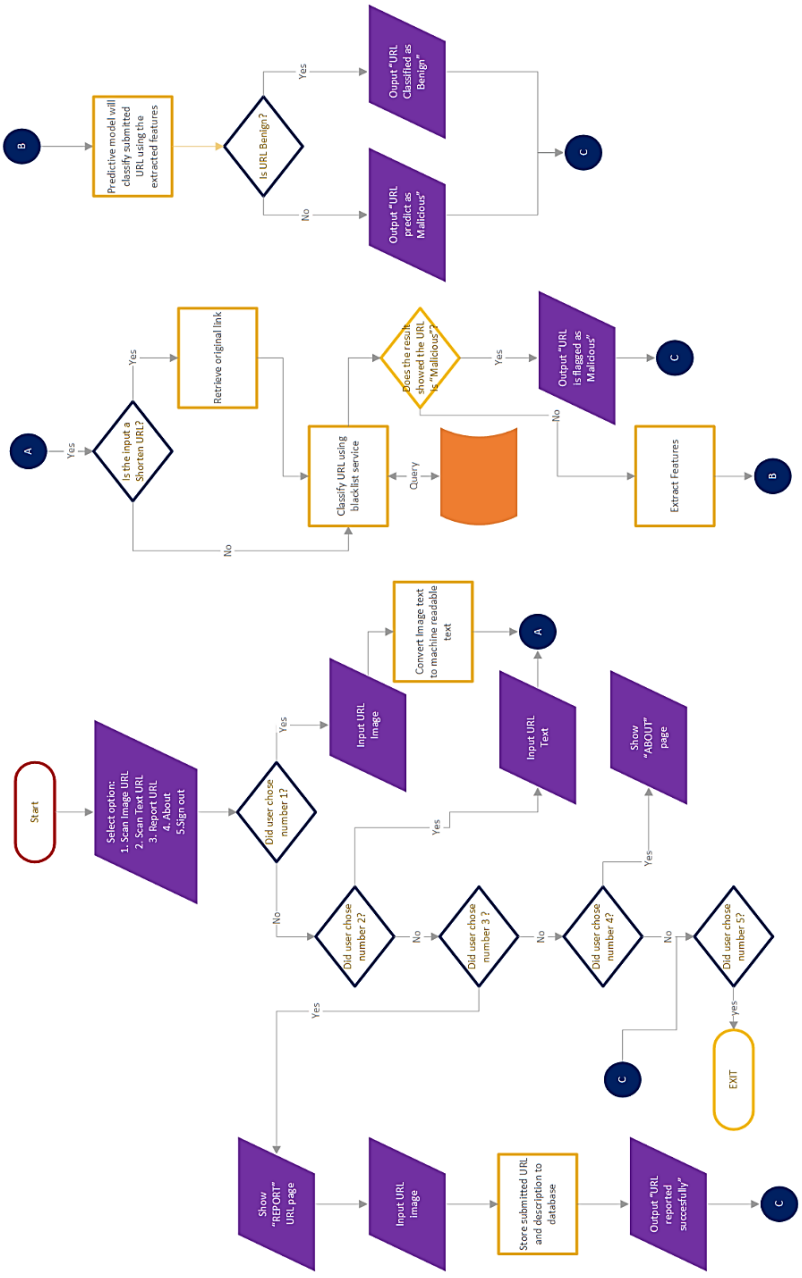


Figure 19. Mobile Application Flowchart

The flowchart shown in figure 19 shows the available functions and the start-to-end processes of the mobile application. One example of its processes is the following: (1) user will first choose to take a picture or upload an image of the URL. (2) The URL image data will be converted into machine-readable text. (3) Detected shorten URLs will be unshortened. (4) The blocklist service will classify the URL by comparing it to labeled URLs from its database. (4) If the result is malicious, the mobile app will display the result. If the result is unknown or benign, the URL features will be extracted and analyzed by the trained classification model. (5)Afterwards, The classification model will predict the label of the URL, whether the URL is malicious or benign. (6) Lastly, the result will be displayed, and the user can exit the app or scan again.

- Web-Based Monitoring System Flowchart

  The Web-based system flowchart is shown in Figure 20.

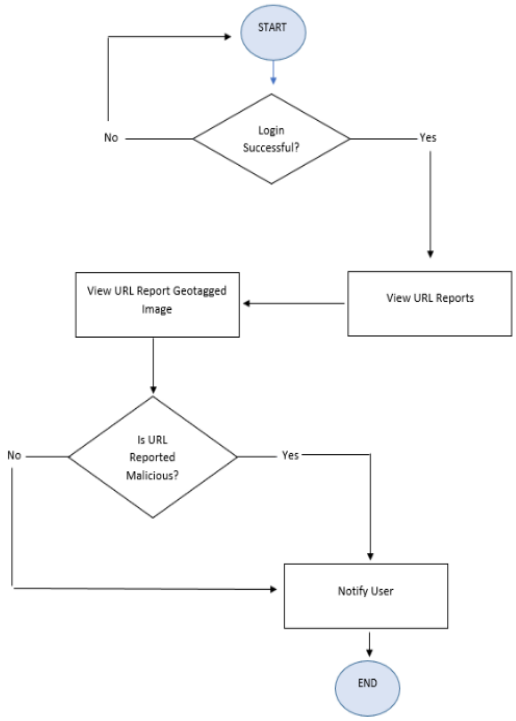


Figure 20. Web-based Monitoring System Flowchart

As shown in the web-based monitoring flowchart in figure 20, the user admin will first log in to his account using his username and password. Once the admin successfully login, the admin can view the URL reports and the summary of the reports in graphical representations. The admin can also view the URL reports in map view, viewing the geotagged image. Moreover, the user can mark the reported URL as malicious or benign and then notify the user by sending an automated email using the user's provided email address.

The deliverables during and after this phase are UI/UX prototype for mobile and web, system architecture, Web monitoring ERD and data dictionary.

Phase 3: Construction

In this phase, the activities the researchers performed are :

A. rapid development of mobile application;
B. rapid development of web-based monitoring system;
C. creation of classification model;
D. mobile and web application unit testing;
E. mobile application accuracy testing; and
F. feedback gathering/usability testing.

The following sections explain the activities mentioned above in more detail

A. Rapid development of mobile application

To develop the mobile application, the researchers used Android Studio as an IDE and Java as the programming language.

- Front end

  Figure 21- 38 shows the front end of the mobile application.

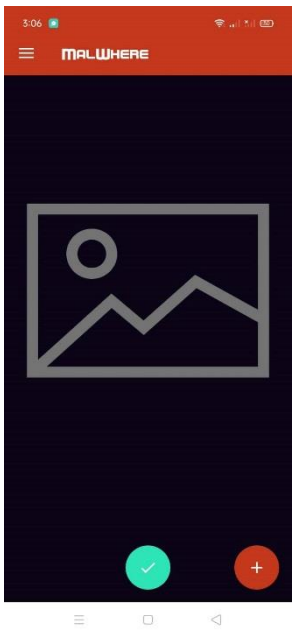


Figure 22. Main Screen

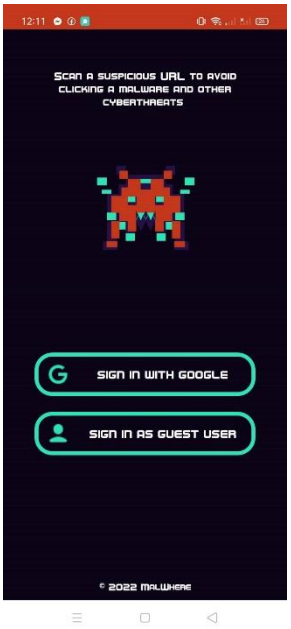


Figure 21. Sign-in

Figure 23. Splash Screen

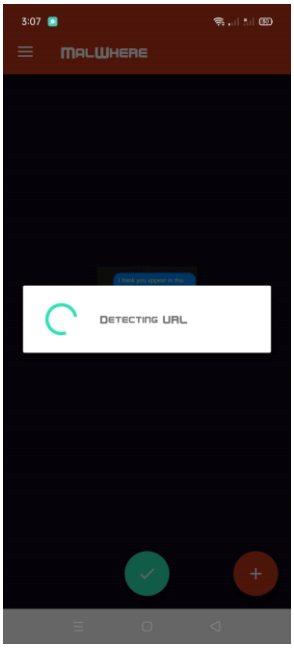


Figure 26. Detect URL from text

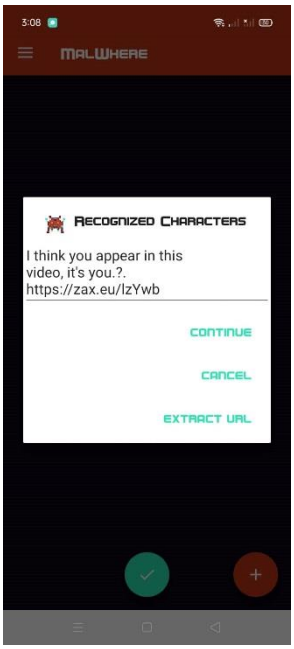


Figure 25. Recognize text from image

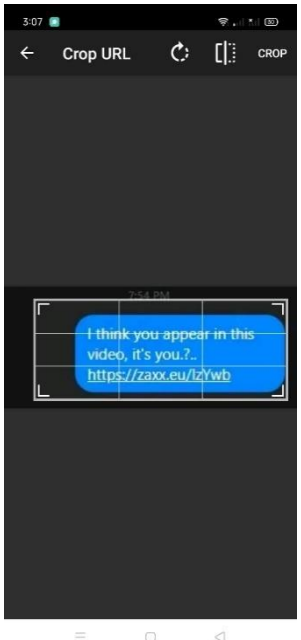


Figure 24. Crop Image URL

Figure 29. Report sent success pop-up message

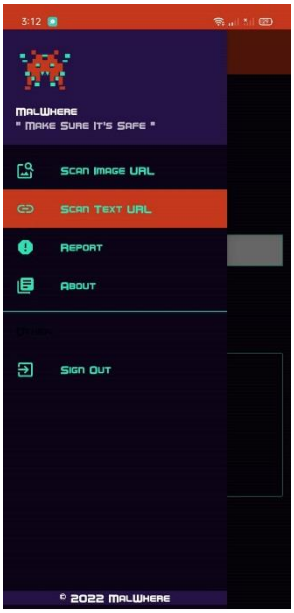


Figure 28. Navigation menu drawer

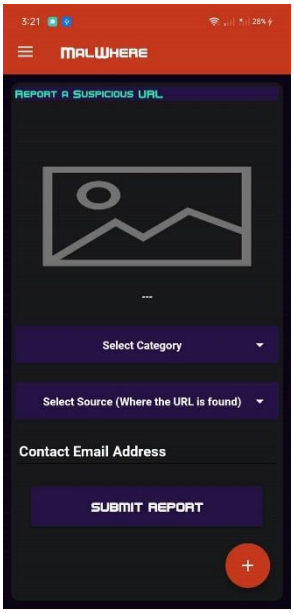


Figure 27. Report URL screen

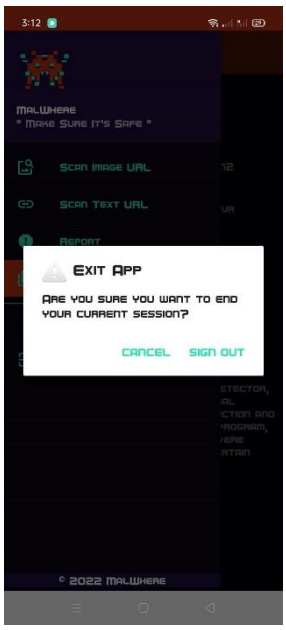


Figure 32. Sign-out/ Exit confirmation message

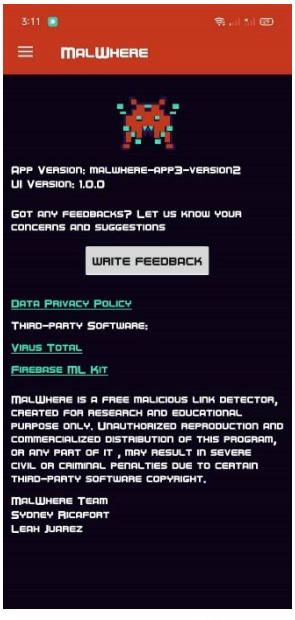


Figure 31. About Screen

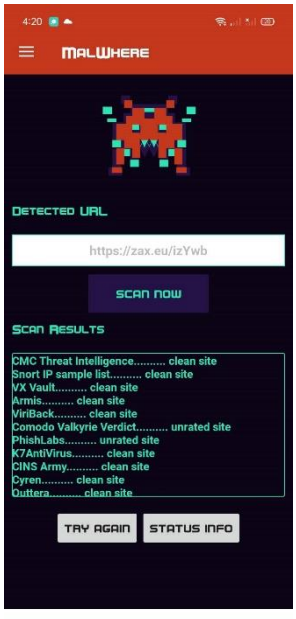


Figure 30. Scan Results

Figure 35. URL Classification pop-up message for benign URL

Figure 34. URL Classification pop-up message for malicious URL

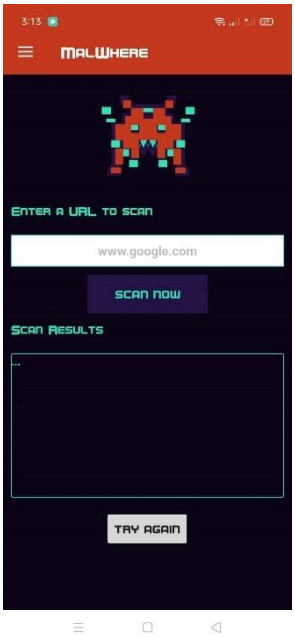


Figure 33. URL Classifying Screen

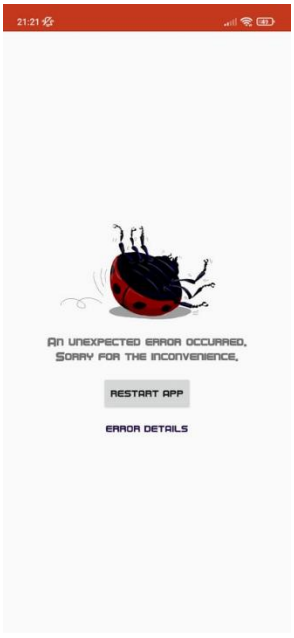


Figure 36. Error Screen

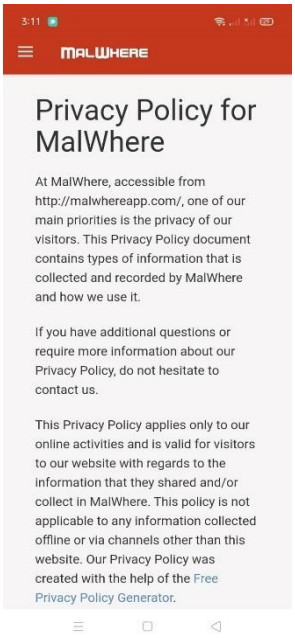


Figure 37. Data Privacy Screen

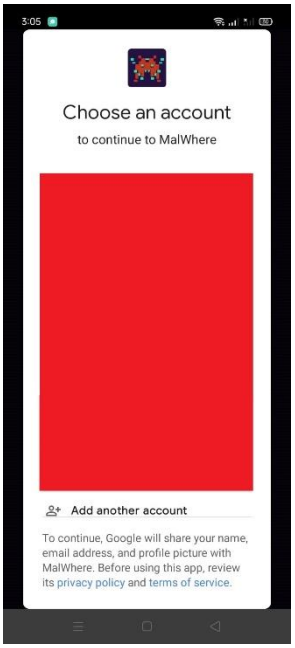


Figure 38. Sign-in with Google SSO

- Back end

  Figure 39 shows the back-end source code snippet of the mobile application.

```
private void extractLongUrl(String urlShorten){

    RequestQueue queue = Volley.newRequestQueue( context: this);
    String myReq = "https://unshort.herokuapp.com/api/?url=" + urlShorten;

    // prepare the Request
    JsonObjectRequest getRequest = new JsonObjectRequest(Request.Method.GET, myReq,  jsonRequest: null,
            new Response.Listener<JSONObject>()
            {
                @Override
                public void onResponse(JSONObject response) {
                    // display response
                    try {
                        //assign the long url to setter and getter
                        new Adapter().setDetected_URL(response.getString( name: "longUrl"));
                        //open new activity scan class
                        startActivity(new Intent( packageContext: MainActivity.this,ScanUrl.class));
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            },
            new Response.ErrorListener()
            {
                @Override
                public void onErrorResponse(VolleyError error) {
                    Log.d( tag: "Error.Response", error.toString());
                }
            }
    );

    // add it to the RequestQueue
    queue.add(getRequest);
}
```

Figure 39.MalWhere app source code snippet

B. Rapid development of web-based monitoring system

For the development of the web monitoring system, the researchers used Visual Studio Code as an IDE, and HTML; CSS; PHP; JavaScript; MySQL; bootstrap; jQuery; and Ajax were used as the programming languages and frameworks.

- Front End

  Figure 40-49 presents the front-end of the Web-based monitoring system.

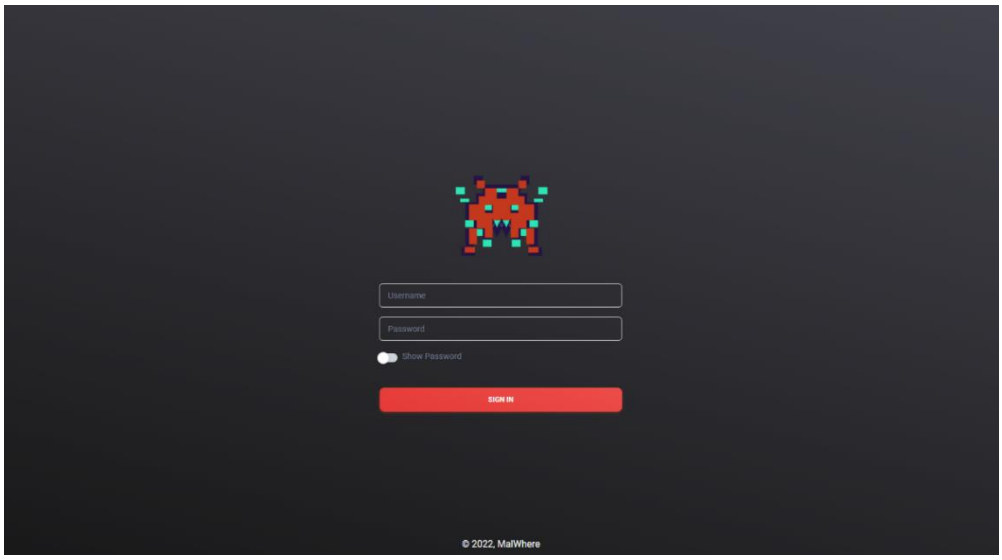


Figure 40. Web Monitoring -Sign-in page

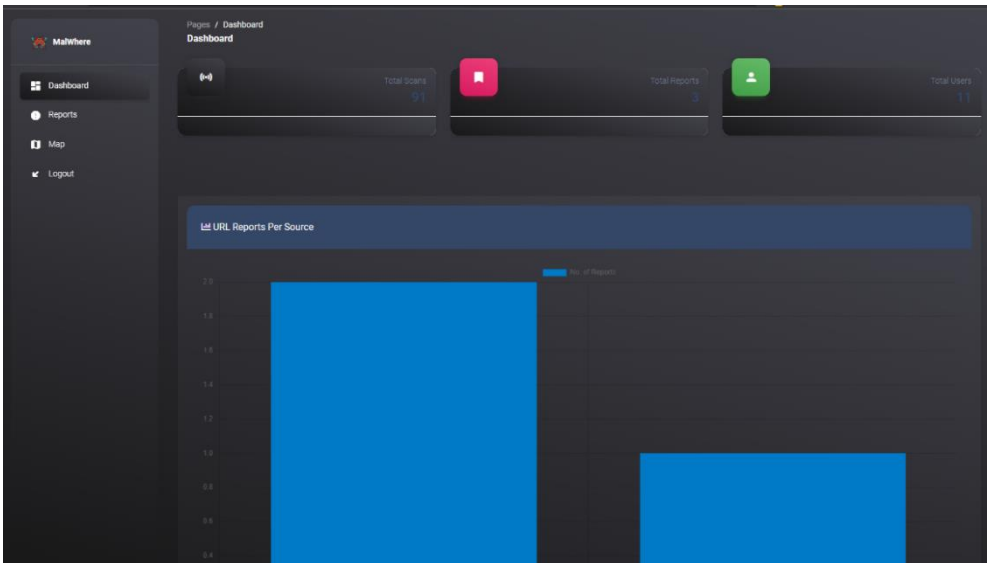


Figure 41. Web monitoring-dashboard

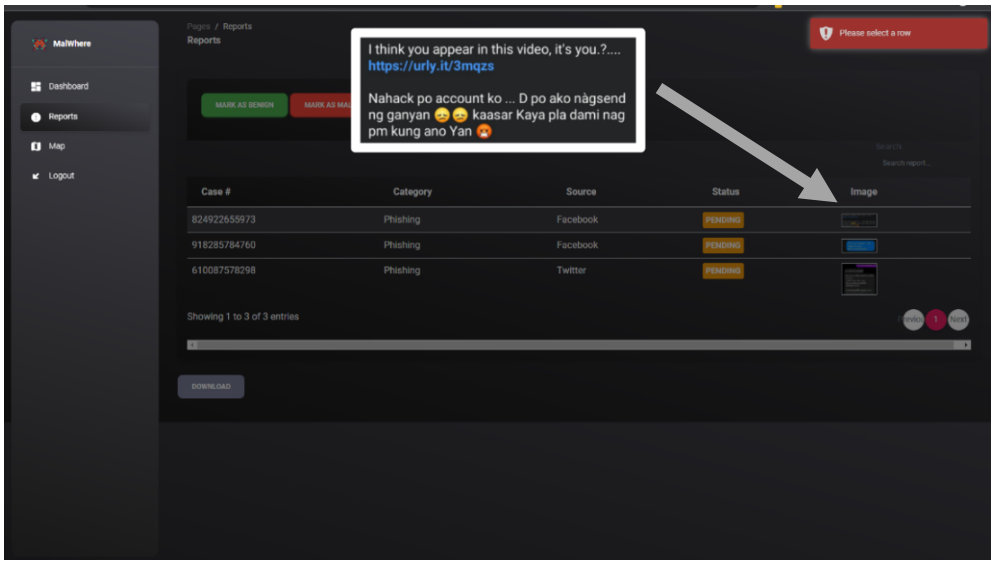


Figure 42. Web report monitoring page-picture view

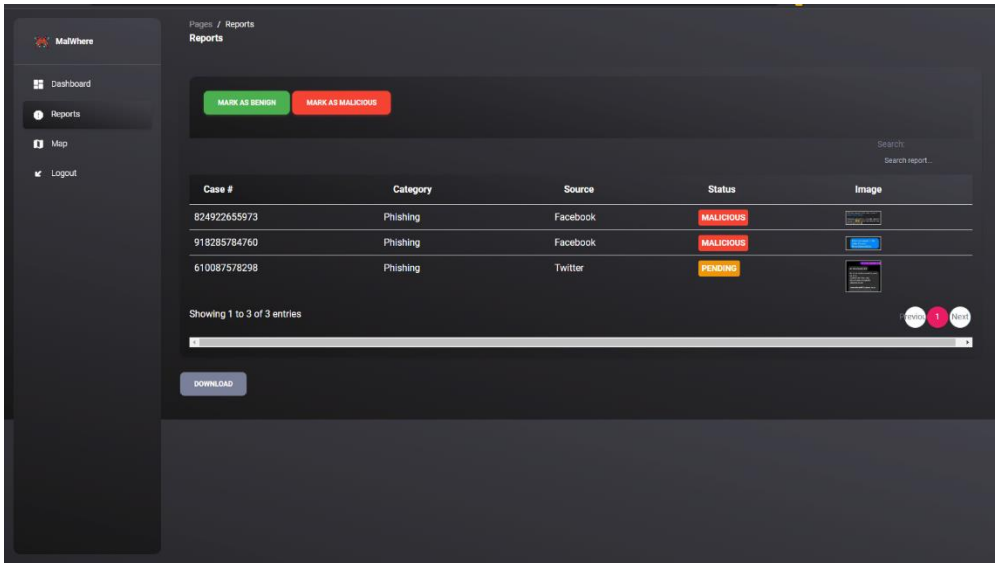


Figure 43. Web report monitoring page

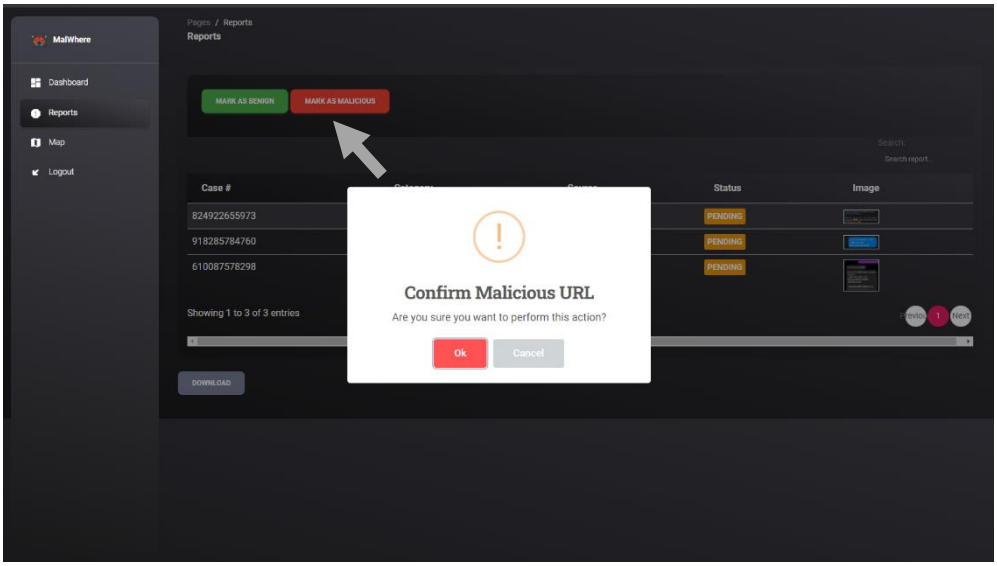


Figure 44. Web report monitoring page-classify user reported URL

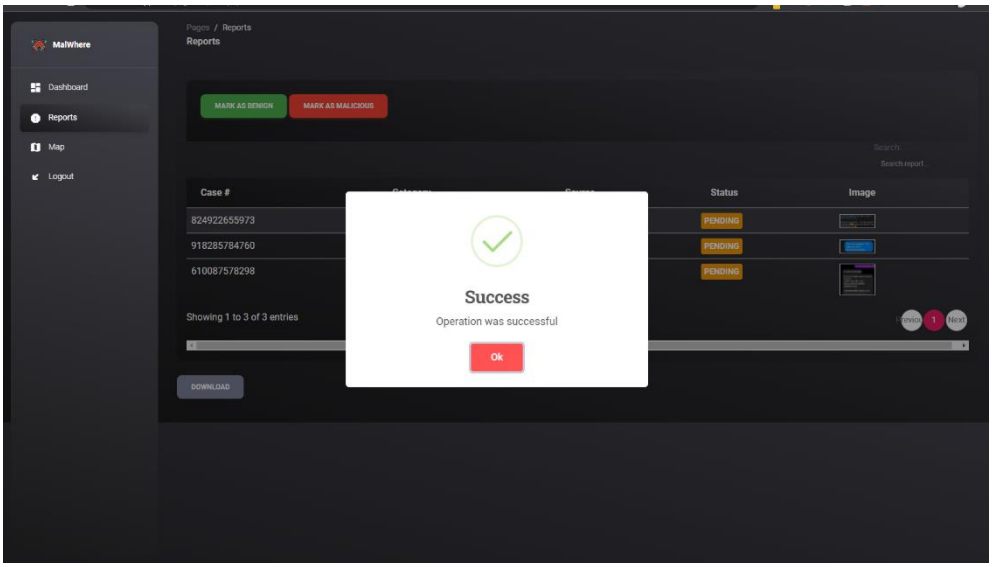


Figure 45. Web report monitoring page-success message for manual classification

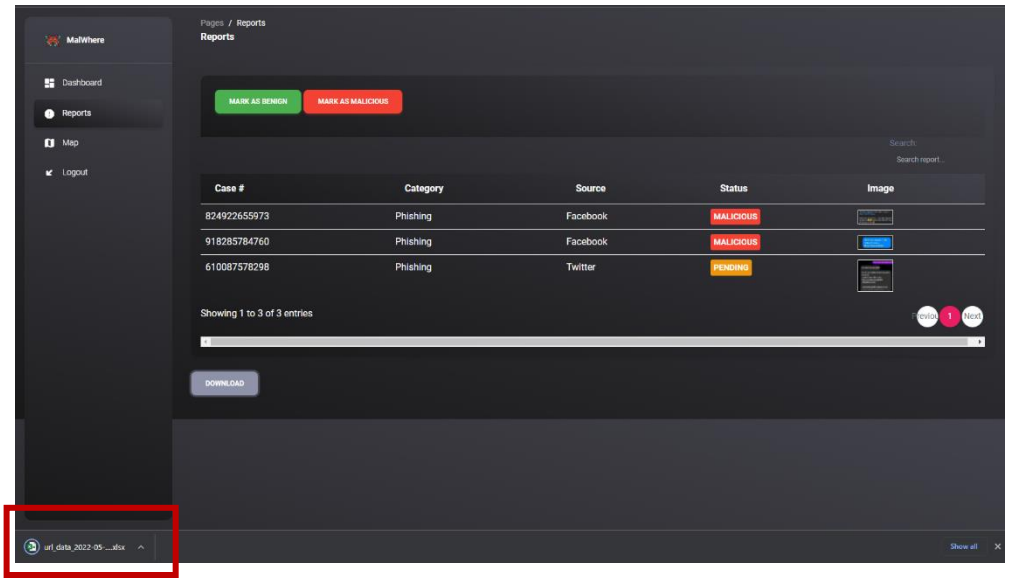


Figure 46. Web report monitoring page -download report

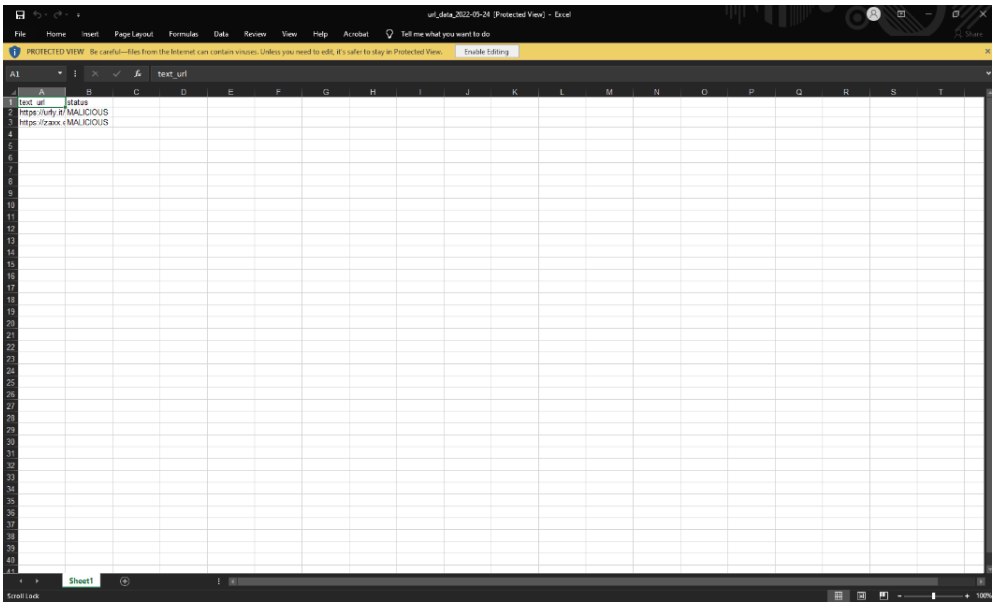


Figure 47. Web report monitoring page-downloaded report in excel file

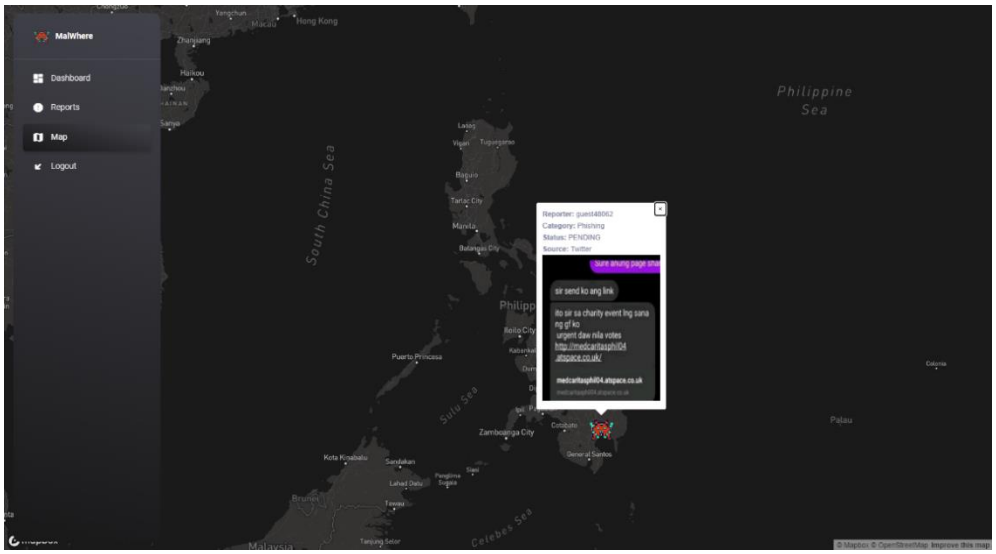


Figure 48. Web monitoring map page

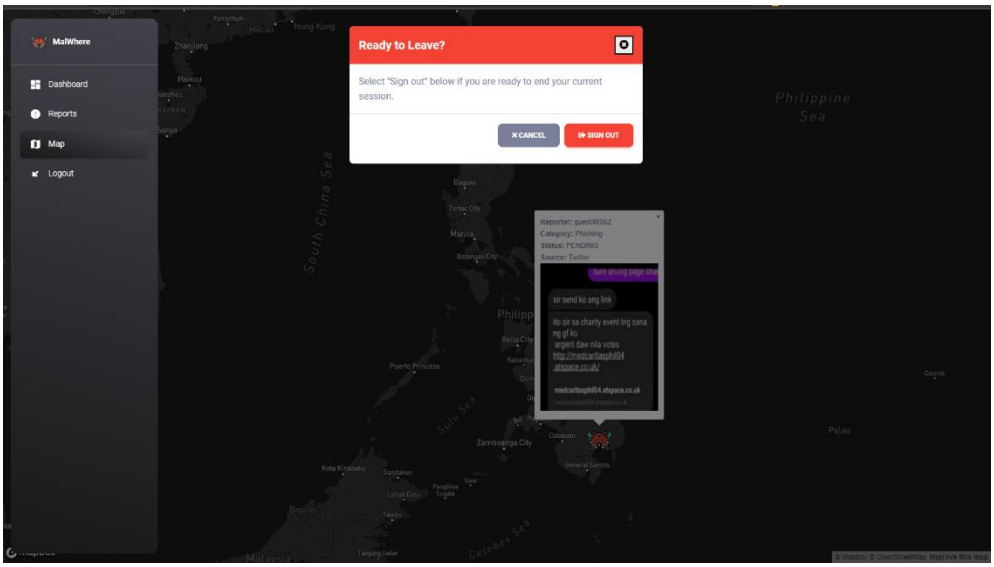


Figure 49.Web logout/sign- out

- Back end

  Figure 50 shows a back-end code snippet of the web-based monitoring system

```
        <tr>
            <td style="display:none"><?= $report['id'];?></td>
            <td><?= $report['caseNum']; ?></td>
            <td><?= $report['category_name']; ?></td>
            <td><?= $report['url_source']; ?></td>
            <td><span id="<?= $report['status'] ?>"><?= $report['status'] ?></span></td>
            <td style="display:none"><?= $report['email']; ?></td>
            <td>
            <a href="#" class="pop">
            <img id="img" data-toggle="tooltip" title="View image" class="border " width="60px" src="
            </a>
            </td>
            <td style="display:none"><?= $report['text_url']; ?></td>
            </tr>
            <?php endforeach; ?>
        </tbody>
    </table>
</div>
</div><br>
<form method="post"><button data-toggle="tooltip" title="Download report" class="btn btn-secondary fl
    Download</button></form>
```

Figure 50. Web monitoring source code snippet

C. Creation of classification model

In creating the classification model, these are the steps that the researchers performed:

a) Create datasets

For extracting URLs and creating URL datasets, the researchers performed the following successively:

1. Download Malicious_n_Non-Malicious URL dataset, malicious URL dataset, and SMS Collection Dataset from the Kaggle platform.
2. Extract and select random URLs from Malicious_n_Non-Malicious and SMS Collection Dataset and insert it to the Malicious URL dataset using excel.

3. Sort URLs by label as benign, phishing, malware, spam, and defacement URLs.
4. Remove URL duplicates.
5. Remove URLs that do not have a protocol.
6. Save the sorted URLs on separate datasets by label in .csv format.
7. Create 2 temporary datasets where one contains 15,000 benign URLs and the other contains 15,000 malicious URLs and save them in .csv format.
8. Check the benign URLs with VirusTotal to check if it is established reliability.
9. Create the 1st official dataset with two columns, then name it as “url” and “label” to be used for training and testing the 1st classification model.
10. Select 10,505 benign URLs and 10,505 malicious URLs from the created 2 temporary dataset (from step 7) to the 1st official dataset. Then rename all non-benign labels (e.g., spam) as “malicious” (now a balance dataset with 21010 URLs is created).
11. Randomized URLs and saved the dataset in .csv format.
12. Create 2nd dataset by repeating step 9-11 but excluding the already selected URLs from 1st dataset, and only select 4,025 benign and 4,025 malicious URLs this time (now a balance dataset with 8,050 URLs is created).
13. Removed 20 benign and 20 malicious URLs in the 2nd dataset and replaced it with other 20 benign and 20 malicious URLs collected through manual collection.

14. In total there were only 29,060 URLs that were used in training and testing the classification model; the unused URLs from the temporary datasets were used as reserved.

The first created dataset will be used for training and testing the 1st classification model and the 2nd dataset will be used for training and testing the 2nd classification model.

b) Select features

This activity is connected to Phase 2 User Design, where the initial features selection had taken place.

Feature correlations are expressed from positive (+) to negative (-) range. A positive correlation indicates that both variables change in the same direction. A negative correlation indicates that the variables are changing in opposite directions. There is no association between the variables if the correlation is 0 [82]. High positive correlated features are removed to save storage and enhance the speed of the classification model.

After the first model was trained and tested, high correlated features were removed based on the feature correlation result. Afterwards the list of selected features was updated. Figure 51 shows the Python code snippet on how the researchers determined what high positive features they should remove.

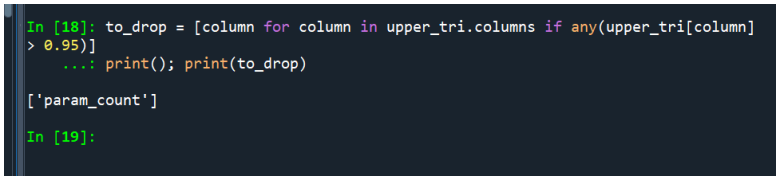


Figure 51. Show classification model features to remove code snippet

In this project study, relating to figure 51, the correlation score that was set as a qualification for a feature to be removed is a feature which has a correlation score of above 0.95. The “param_count” feature is the only feature who fit the describe condition. The accuracy of the classification model before the removal of “param_count” feature, was still the same (i.e., 99.486008% or 99.48%) after the feature removal. The final list of features can be seen in Table 6, in Phase 2 User Design. This final list is the same list of features used to train and test the 2nd model which is the final model.

c) Create, train, and test the classification model.

In this activity, the goal was to create a high-accurate classification model. In pursuit of the goal, the researchers conducted one iteration for model creation. As mentioned in the preceding section. There are two datasets created and were used to create the two classification models.

In creating, training and testing of the classification model, scikit-learn, pandas, xgboost, numpy, whois, math, datetime, time, re, urllib, seaborn, tld, itertools, matplotlib, and joblib libraries, were used.

The first model has a dataset with 21,010 URLs (i.e., 10,005 benign & 10,005 malicious), divided for training and testing. The researchers split the dataset using the “train_test_split” python function, with a 75:25 train-test ratio, to evaluate the accuracy of the first classification model.

XGBoost has many hyperparameters that can be set and tuned in through a series of experiments to avoid overfitting and to make a model achieve the optimal performance; however, in this research, the researchers only used the default hyperparameters' values. Figure 52 shows how the 1st model was created, and its accuracy using the XGBoost.

```
In [11]:
    ...:
    ...: X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size=0.25,shuffle=True, random_state=45)

In [12]:
    ...: model = xgb.XGBClassifier()
    ...: model.fit(X_train,y_train)
    ...: y_pred = model.predict(X_test)

In [13]:
    ...:
    ...: print(classification_report(y_test,y_pred))
    ...:
    ...: #99.49% 39features
    ...: #Accuracy lang ang mugawas
    ...: score = metrics.accuracy_score(y_test, y_pred)
    ...: print("Testing accuracy:   {0:.4f} %".format( score *100))
    ...:
    ...:
    ...: #training Accuracy
    ...: print("Training Accuracy:  {0:.4f} %".format(model.score(X_train, y_train)*100))
    ...: #print("Test score: {0:.5f} %".format(100 * score))
    ...:
    ...: print(model.objective)
    ...: print(model)
    ...:
    ...:
    ...:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      2626
           1       1.00      0.99      0.99      2627

    accuracy                           0.99      5253
   macro avg       0.99      0.99      0.99      5253
weighted avg       0.99      0.99      0.99      5253

Testing accuracy:   99.4860 %
Training Accuracy:  100.0000 %
binary:logistic
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

Figure 52. First classification model creation code snippet

As shown in figure 52, the 1st model has an average precision rate of 99%, average recall rate of 99%, average f1-score of 0.99, training accuracy of 100%, and a testing accuracy of 99.47%.

Now in the 2nd model, the researchers used the 2nd dataset, which has 8,050 URLs (i.e., benign 4,025 & malicious 4,025). Figure 53 shows how the 2nd model was created and its accuracy.

```
In [20]: X_train2_s, X_test2_s, y_train2_s, y_test2_s = train_test_split(X2_s, y2_s,
stratify=y2_s, test_size=0.25,shuffle=True, random_state=45)
    ...:
    ...: import joblib
    ...: #load saved model
    ...: model4 = joblib.load("malwhere_modelv.4-21010k(dump).json")
    ...:
    ...: model4_2_S = xgb.XGBClassifier( xgb_model='model4')
    ...: #prin (model)
    ...: model4_2_S.fit(X_train2_s,y_train2_s)
    ...: y_pred2_s = model4_2_S.predict(X_test2_s)
    ...:
    ...: print(classification_report(y_test2_s,y_pred2_s))
    ...:
    ...: #Accuracy lang ang mugawas
    ...: score = metrics.accuracy_score(y_test2_s, y_pred2_s)
    ...: print("Testing accuracy:   {0:.4f} %".format( score *100))
    ...:
    ...: #training Accuracy
    ...: print("Training Accuracy:  {0:.4f} %".format(model4_2_S.score(X_train2_s,
y_train2_s)*100))
    ...: #print("Test score: {0:.5f} %".format(100 * score))
    ...:
    ...: print(model4_2_S.objective)
    ...: print(model4_2_S)
    ...:
    ...:
    ...:
[14:46:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/learner.cc:
627:
Parameters: { "xgb_model" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.


              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1006
           1       1.00      0.99      0.99      1007

    accuracy                           0.99      2013
   macro avg       0.99      0.99      0.99      2013
weighted avg       0.99      0.99      0.99      2013

Testing accuracy:   99.2052 %
Training Accuracy:  100.0000 %
binary:logistic
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

Figure 53. Second classification model creation code snippet

The dataset division and process were the same, except in the 2nd model, which was created including the created first model. This technique is the ensemble method of XGBoost, which is developed for other model/tress to avoid committing the same errors that the first model/trees committed which results in the creation of a more reliable and improved model which will then decide the final prediction. In comparison with other classifiers like Random forest,

its ensemble method train decision trees in parallel and decides the final prediction by majority voting.

As shown in figure 53, the 2nd model has an average precision rate of 99%, average recall rate of 99%, average f1-score of 0.99, training accuracy of 100%, and a testing accuracy of 99.21%.

In this experiment, even though the 2nd model is expected to be more accurate, as observed, the 1st model has a higher accuracy rate (i.e.,99.47%) compared to the 2nd model (i.e., 99.21%). Every after, a model was tested; The model was saved as JSON file using the python joblib library. Because the result of the 2nd model was not what was expected, another series of tests was conducted to know if the training of the 2nd model was successful and to know their difference in prediction by comparing the prediction result of the 1st model and 2nd using the same set of data. The researchers conducted another test using new sample data. Figure 54 shows the summary of the prediction results.

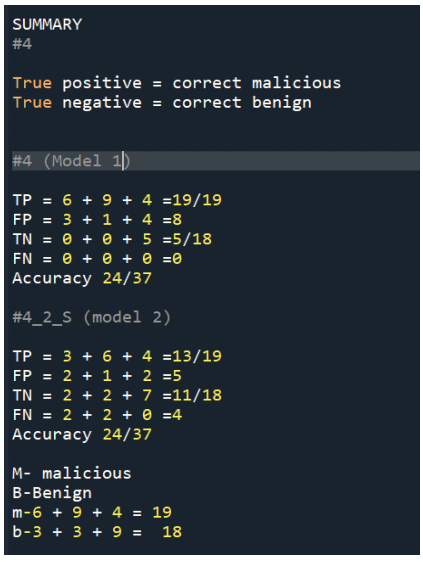
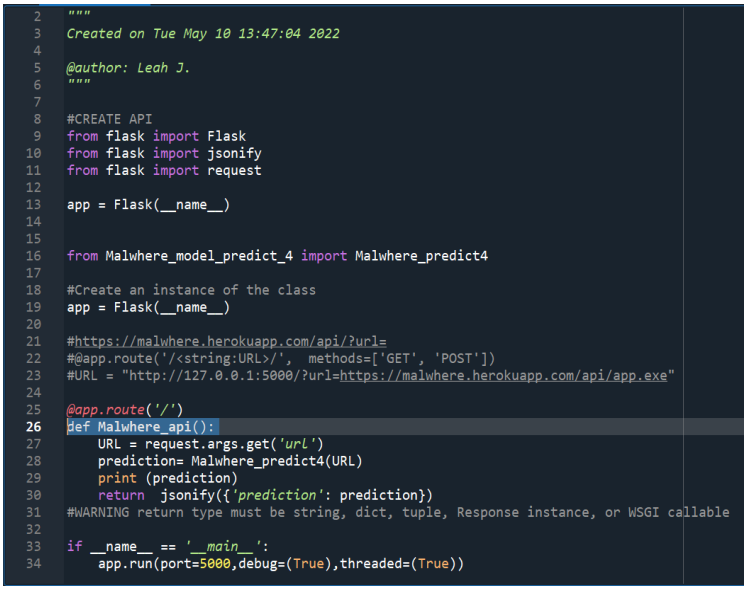


Figure 54. Prediction result

As shown in figure 54, both have the same frequency of correct predictions (i.e., 24/37) conducted through 3 subsequent experiments. Based on the results, model 1 predicts malicious URLs with high accuracy and has a high tendency to predict benign URLs as malicious, which the researchers saw as a weakness; on the other hand, model 2 predicts both malicious and benign with high accuracy. With this difference in prediction pattern, the training of the 2nd model, which is also the final model, was confirmed to be successful. Furthermore, a conclusion that a reliable classification model would be integrated into the mobile application was made.

For the integration of the classification model into the mobile application. The researchers developed it as a REST API that receives an HTTP GET request and throws a JSON response. The API was made using Flask and was deployed using Heroku. Flask is a web framework and a Python module that allows developers to create web apps easily [83]. Figure 55 shows the flask code, and figure 56 shows the deployed API displaying its JSON response.



```
"""
Created on Tue May 10 13:47:04 2022

@author: Leah J.
"""

#CREATE API
from flask import Flask
from flask import jsonify
from flask import request

app = Flask(__name__)


from Malwhere_model_predict_4 import Malwhere_predict4

#Create an instance of the class
app = Flask(__name__)

#https://malwhere.herokuapp.com/api/?url=
#@app.route('/<string:URL>/',  methods=['GET', 'POST'])
#URL = "http://127.0.0.1:5000/?url=https://malwhere.herokuapp.com/api/app.exe"

@app.route('/')
def Malwhere_api():
    URL = request.args.get('url')
    prediction= Malwhere_predict4(URL)
    print (prediction)
    return  jsonify({'prediction': prediction})
#WARNING return type must be string, dict, tuple, Response instance, or WSGI callable

if __name__ == '__main__':
    app.run(port=5000,debug=(True),threaded=(True))
```

Figure 55. Flask code snippet

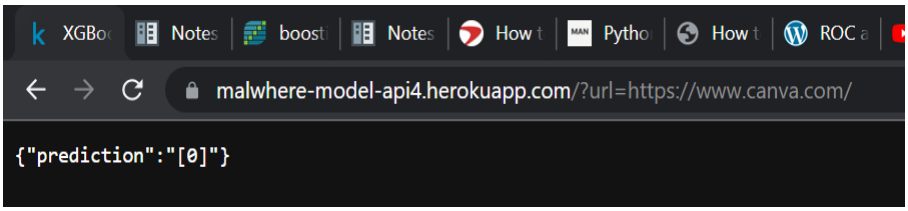


Figure 56. API deployed in Heroku screenshot

D. Mobile and web application unit testing

User and System requirements are tested and recorded in Unit Test Report, seen in Appendix E.

E. Mobile application accuracy testing

Mobile application accuracy testing is presented in the next chapter, chapter 4 results, and discussion.

F. Feedback gathering.

Any system's usability is a crucial factor in its overall evaluation. Hence, UX designers should assess the usability of each new system they suggest and strive to make it simple to use for end-users. In pursuit of assessing the usability of the developed mobile application, the researchers invited participants for the usability study through word of mouth, Facebook, and messenger groups. Participants were requested to download and install the MalWhere application, utilize it, and complete the online survey in google forms. There were 69 respondents who answered the survey. The prepared survey has 2 sections: feedback and usability testing.

The usability survey questions adopted the ten standard questions of the System Usability Scale (SUS) created by John Brooke. SUS statements cover different aspects of system usability, such as the need for support, training, and complexity; hence it has a high level of face

validity for measuring the usability of a system. [73]. Other researchers also use SUS to test the usability of their deployed systems [22].

For ethical consideration, data collection consent is included in the first section of the distributed google forms pursuant to the Republic Act No. 10173 or Data Privacy Act of 2012.

The deliverables in this phase are the developed mobile application, web application and mobile application feedback and usability survey.

Phase 4: Cutover

Cutover is the last phase. The deliverables in this phase are the developed and deployed mobile and web applications with system changes based on user feedback.

CHAPTER 4
RESULTS AND DISCUSSIONS

This chapter shows the results and discussion of the conducted project study. The results presented are in accordance with the objectives of the study.

Extract text from an image data

To give the mobile application the ability to extract texts from image data, the researchers integrated it with Firebase ML kit API. Figure 57 shows the snippet of the code.

```
private void processText(FirebaseVisionText text){
    List<FirebaseVisionText.TextBlock> blocks = text.getTextBlocks();

    if(blocks.size() ==0){
        Snacky.builder()
                .setText("No Text")
                .setIcon(R.drawable.ic_error_outline_black_24dp)
                .info()
                .show();

        return;
    }

    recognizedText = new StringBuilder();

    for (FirebaseVisionText.TextBlock block: text.getTextBlocks()){
         recognizedText.append(block.getText());
    }
    recogTextDialog();

}
```

Figure 57. Extract text from an image code snippet

Based on the respondents' feedback, they have confirmed that the mobile app can extract text from screenshots and camera captured images. However, extracted text are observed to be often inaccurate especially when it was extracted from an image with a lot of image noise which commonly are text images captured along with a device screen or monitor, by a camera.

Extract URL from recognized text

After extracting texts from an image, the application should detect and extract the URL among the block of text. The researchers used a URL Detector API to give the mobile application the ability to extract URLs from a block of texts. LinkedIn Security Team develops this API. Aside from extracting URLs, the researchers also used Unshort API, an API that extracts the full URL of a shortened URL so that the actual URL will be used in classification and also for the researchers not to conduct a further experiment to know what practical features to use in training the classification model, to differentiate malicious shortened URL from benign shortened URL as well as benign long URL. Figure 58 shows the long URL extraction from shorten URL code snippet and UI.

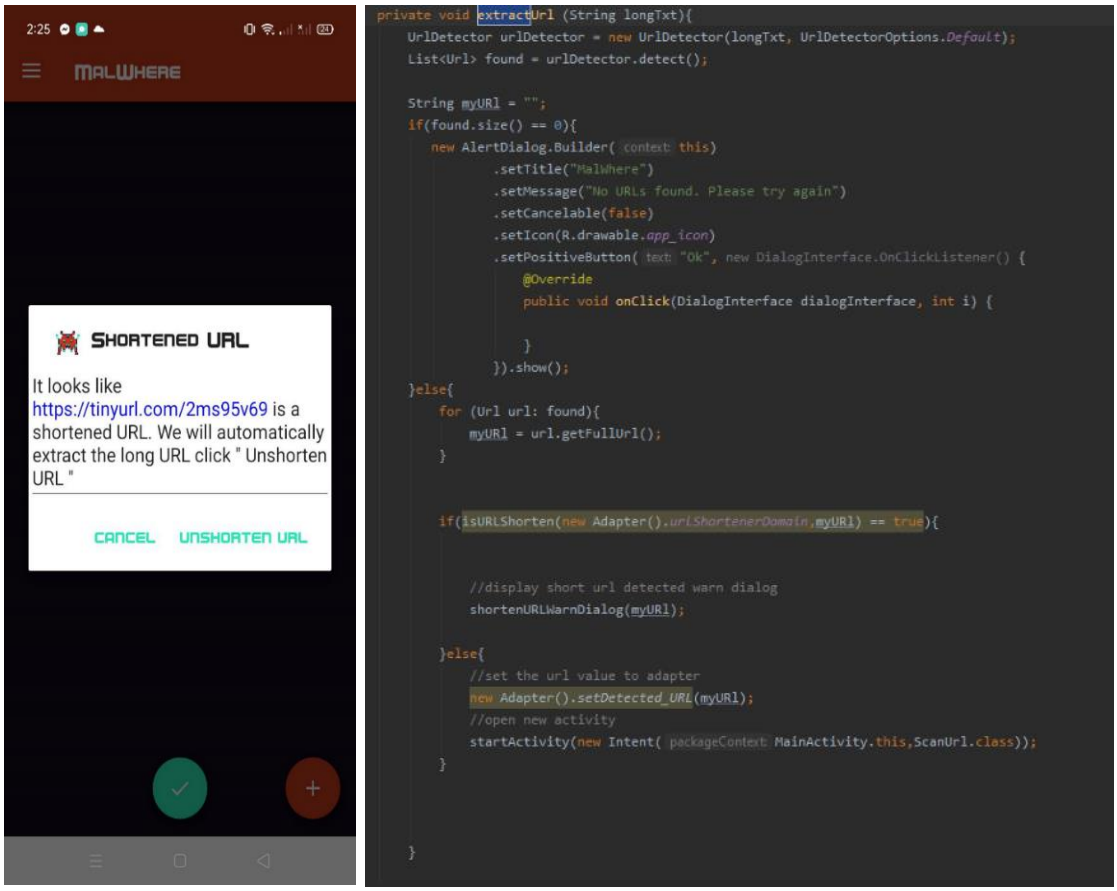


Figure 58. Extract the long URL of a short URL code snippet and UI

This is the researchers' solution on how to classify shortened URLs effectively. However, the utilized API that unshortens a shortened URL does not support all existing shortening services.

Based on the respondents' feedback, the URL extractor when extracting URL from a block of text sometimes does not have accurate results. This inaccurate result will invalidate the classification result due to incorrect input. To deal with this issue, the researchers made URL extraction optional. Now users can choose not to use the URL extractor after the mobile application recognized the texts from an image. Furthermore, the researchers designed the mobile application to make the extracted texts editable, so that the user can correct or remove wrong text or characters.Furthermore, shortened URLs that has shortener services that is not supported by the mobile application is proceeded to be classified in its shortened form.

Automatically classify whether the URL is benign or malicious

In order to help internet users automatically classify whether a URL is benign or malicious, two different URL classification approaches are combined and utilized in the mobile application. These two are the backlisting approach and the machine learning approach. The blocklist approach was implemented using Virus Total, and the machine learning approach was implemented using the created classification model. The classification model was trained with XGBoost, a supervised machine learning classifier. The two are combined by making the blocklist service the primary classifier and the classification model the secondary classifier. This means that Virus Total first classifies the URL; if the result is benign or unknown, then the classification model will predict its classification. Including to predict URLs classified as benign by the blocklist service is the intervention of the researchers to avoid or lessen the false-negative rate. Figure 59 shows the pop-up dialog of a URL classified as malicious and the blocklist service in combination with classification model implementation code snippet.

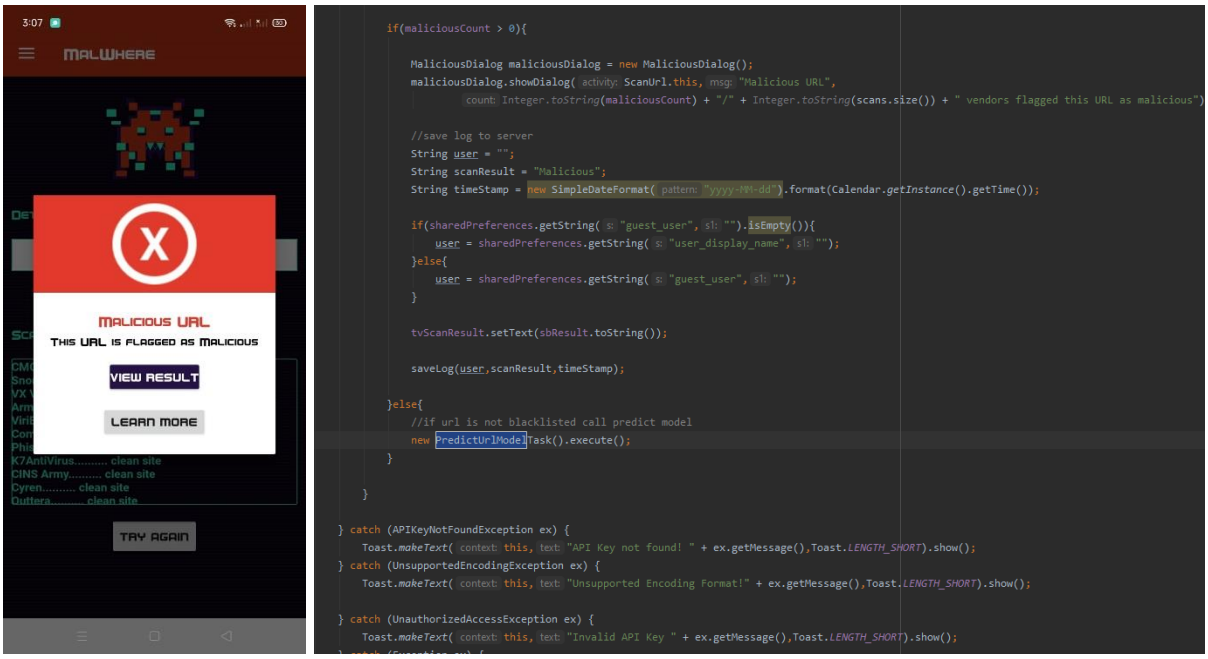


Figure 59. Pop-up dialog of a URL classified as malicious and the blocklist service in combination with classification model implementation code snippet.

Based on the respondents' feedback, the mobile app classifies URL whether it is benign or malicious.

Monitor the reported URLs and extract the validated reported URLs for classification model update and improvement

URL report is a feature that supports and enhances the classification accuracy of the mobile application. Though the classification ability of the mobile application has a high accuracy rate it still needs improvement to lessen incorrect predictions; this led the researchers to make a feature in the mobile application to report URLs that are classified incorrectly. Users who have an idea of the actual classification of the URL can report the URL and determine its classification. The sent report will be stored in the database and will reflect on the web-based monitoring system. Furthermore, from there, the admin will monitor the sent URL reports and manually investigate the sent URL through a different method such as manual lookup and using other trusted blocklist services.

After the investigation, the admin will confirm the URL label and the system will automatically send an email message to the reporter of the URL. The URLs with confirmed labels will be stored in the database and can be downloaded so that it can be utilized as a new dataset to train and update the classification model for improvement. Figure 60 shows the automated response of the MalWhere web-based monitoring system after the URLs' classification is confirmed.

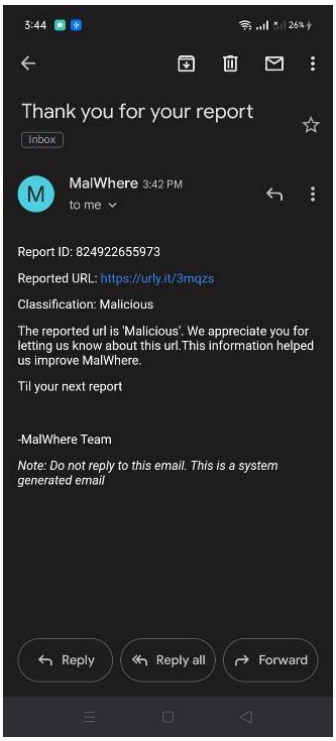


Figure 60. Automated email response

Based on the respondents' feedback, they can send a report. Meanwhile, based on the unit testing, the web-based monitoring system can receive, display and let the admin confirm the actual classification of the reported URL after the admin's investigation. In addition, the automatic email response message regarding the actual classification of the reported URL is confirmed to be functioning. When the URL classification confirm button is clicked, this automatic email reply is sent to the reporter.

Test the accuracy of the mobile application.

The test was conducted on three different input methods which are image upload, camera capture, and copy & paste. Furthermore, the researchers observed the disadvantage of the utilized text extractor API, which often outputs inaccurate text extraction results, hence the researchers made sure that the inputs are not invalid by editing the extracted text to reflect the raw test URL in the input field to effectively measure the classification accuracy of the mobile application. Afterward, the performance of the mobile application is computed by using precision, recall, f1-score, and accuracy metrics with the help of the confusion matrix.

Test Environment #1: Image Upload

A sample uploaded screenshot image is shown in figure 61.

Figure 61. Uploaded image sample

Table 7 shows the summary of the test cases that were conducted to test the classification accuracy of the mobile application through image upload input method.

Table 7. Image upload test case

Test case no.	Method	URL	Result	Accurate
1	Image Upload	http://www.vouch4me.com/etlp/dining.asp	Malicious	Yes
2	Image Upload	http//www.gr8prizes.com 08715705022	Malicious	Yes
…				
40	Image Upload	http://88.218.17.197/Seijin.i586	Malicious	Yes

Out of 120 test cases, 40 were conducted through image capture. Based on these test case results; the mobile app can obtain a classification accuracy rate of 90% (36/40).

Test Environment #2: Camera Capture

A sample captured image is shown in figure 62 .

Figure 62 Captured image sample

Table 8 shows the summary of the test cases that were conducted to test the classification accuracy of the mobile application through camera capture input method.

Table 8. Capture image test case

Test case no.	Method	URL	Result	Accurate
1	Camera Capture	http://greatplainschurch.org/goehner-campus/sermons.html?start=609	Malicious	Yes
2	Camera Capture	http://liegen-sitzen.de/sitzen/schlafsofas	Malicious	Yes
…				
40	Camera Capture	https://www.grab.com/sg/download/	Benign	Yes

Out of 120 test cases, 40 were conducted through camera capture. Based on these test case results; the mobile app can obtain a classification accuracy rate of 80% (32/40).

Test Environment #3 : Copy & paste

Table 9 shows the summary of the test cases that were conducted to test the classification accuracy of the mobile application through copy & paste input method.

Table 9. Copy & paste test case

Test case no.	Method	URL	Result	Accurate
1	Copy & paste	http://remins.booking-artist.ro/remax/	Malicious	Yes
2	Copy & paste	http://aaload05.top/downfiles/4.exe	Malicious	Yes
…				
40	Copy & paste	https://dl.acm.org/action/showLogin?redirectUri=%2F	Malicious	No

Out of 120 test cases, 40 were conducted through copy & paste. Based on these test case results; the mobile app can obtain a classification accuracy rate of 92.5% (37/40).

Summary of test results

The summary of the test case results that were conducted using the three different input methods is shown in table 10 and table 11. Table 10 shows the non-normalized Confusion Matrix of the test cases. On the other hand, Table 11 shows the classification report summary, using the precision, recall, f1-score, and accuracy as metrics to measure the performance of the mobile application.

Table 10. Non-Normalized Confusion Matrix

		Actual	
		Malicious (P)	Benign (N)
Predicted	Malicious (P)	TP=59	FP=14
	Benign (N)	FN=1	TN=46

As shown in table 10, the mobile application has a true-positive rate (TPR) of 49%, a false-positive rate (FPR) of 12%, a true-negative rate (TNR) of 38%, and a false-negative rate (FNR) of 1%.

The classification report summary follows how the "classification_report" function of Scikit-learn python library calculates the performance metrics of a model.

Table 11. MalWhere Classification report summary

	precision	recall	f1-score	Support
Malicious	0.81	0.98	0.89	60
Benign	0.98	0.77	0.86	60
accuracy			0.88	120
Macro avg	0.89	0.88	0.87	120
Weighted avg	0.89	0.88	0.87	120

The performance metrics are computed with reference to the Confusion Matrix shown in table 10 and using the following formula:

- Precision (true positive/predicted positive)

  = TP/(TP + FP) [formula for malicious class]

  or

  = TN/(TN+FN) [formula for benign class]

- Recall (true positive/actual positive)

  = TP/(TP + FN) [formula for malicious class]

or

  = TN/(TN+FP) [formula for benign class]

- F1 (harmonic mean of precision and recall)= 2 ∗ (precision ∗ recall) / (precision + recall)
- Accuracy (all correct/all)= (TP + TN)/(TP +TN + FP+ FN)

- Support = number of actual occurrences of the class in the specified dataset.
- Macro Avg = (class score 1 + class score 2+…class score N)/number of different classes
- Weighted Avg= (class score 1*total of class 1+ class score 2* total of class 2 + …class score N * total of class N)/total of all class values.

Overall based on the classification report summary presented in table 10, the MalWhere app can obtain a classification accuracy rate of 88%. Short URLs that were not unshorten contributed to the cause of incorrect prediction.

Test the usability of the mobile application.

To test the usability of the mobile app, the researchers created a usability survey which was distributed online through google forms targeting mobile internet users. Afterward, the system usability score that was given by each student was computed and also the overall average score. The following are the formula used to compute the SUS score and average SUS score.

- Contribution score (for items 1,3,5,7,and 9) = scale position -1
- Contribution score (for items 2,4,6,8 and 10) =5 - scale position
- SUS Score = sum of ten contribution scores / 2.5
- Average SUS score = SUS score/ the total number of respondents.

To calculate the average SUS score, the first thing to do is calculate each contribution score per item, then calculate the SUS score per respondent then finally calculate the average SUS score.

Based on the computation, the usability score for MalWhere is 56.81. Based on the adjective rating scale presented by Bangor et al., MalWhere has an equivalent adjective rating of "ok" [75]. The Adjective rating scale can be seen in Chapter 2, figure 5.

Furthermore, the mobile application's response time, or the duration of how long it takes to display the classification result, is also tested. Based on the respondents' recorded answers, the mobile application mostly has a 3 seconds response time, 1-second minimum response time, and 2 minutes maximum response time.

CHAPTER 5
SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

This chapter contains the summary, conclusions, and recommendations based on the empirical results of the project study.

SUMMARY

The following is the summary of the project study.

- The researchers developed and deployed a mobile and website application to hostinger.ph cloud server hosting service.
- The researchers use and combined blocklist approach and machine learning approach in classifying URLs whether it is benign or malicious. The two approaches are combined by making the blocklist service the primary classifier and the classification model the secondary classifier. This means that Virus Total first classifies the URL. If the result is benign or unknown, then the classification model will predict its classification, whether it is benign or malicious. Including to predict URLs classified as benign by the blocklist service is the intervention of the researchers to avoid or lessen the false-negative rate.
- The researchers created a classification model with a 99% accuracy rate.
- Based on the test results the mobile application has a classification accuracy rate of 88% by combining a blocklist service and the created classification model.
- The mobile application accepts URL through any of the 3 input methods which are image upload, camera capture, and copy & paste.
- The mobile application can extract text from an image using Firebase ML Kit API.
- The mobile application can extract URLs from text using the URL detector library developed by LinkedIn Security Team.

- The mobile application can extract the full URL of a shortened URL using Unshort API developed by Siddiqui Affan .
- The short URLs that was not able to unshorten by the API contributed to the cause of the incorrect prediction of the MalWhere app.

CONCLUSIONS

In this project study, the researchers developed and deployed 3 outputs. A mobile application, a classification model and a web application.

The researchers developed and deployed a mobile application that classifies a URL whether it is benign or malicious. Based on the test results, the mobile application has an accuracy rate of 88% by combining a blocklist service, VirusTotal, and the created classification model, with image upload, camera capture, and copy & paste as its input method. However, shortened URLs that was not able to unshorten by the API, negatively affects the classification accuracy of the mobile application.

URLs can be classified using a blocklist service, however blocklist services has a serious false-negative rate and does not classify unknown URLs from its database, hence the researchers created a classification model. The created classification model can obtain 99% classification accuracy rate, based on test results. It was trained using the XGBoost algorithm with its default hyperparameters and using a total of 39 features, of which 37 are lexical-based, and the other two are host-based.

Furthermore, the researchers also developed and deployed a web application that will help the admin monitor and manage the user reported URLs. Reported URLs will be investigated to confirm its true classification. Validated reported URLs and its labels can be downloaded to use as a new dataset for training and testing of the classification model for improvement. This can result to the mobile application’s increase in classification accuracy rate.

In conclusion, the researchers developed and deployed a mobile application that can detect and classify a URL, whether it is benign or malicious; a classification model that supports the mobile application in predicting the classification of a URL; and a web application that can be used to managed reported URLs and to collect new URLs to use as a new dataset for training and testing the classification model, for improvement.

RECOMMENDATIONS

Based on the results of the study, the researchers recommend the addition of the following features or functions in the system for improvements:

- Real-time detection feature
- Multiple URL recognition feature.
- Tuned in XGBoost hyperparameters when training the classification model to lessen overfitting, reduce model complexity and enhance model performance.
- Utilize and integrate an API/library/technology that helps identifies the responsible party of a malicious URL, such as advertising networks and application developers.
- Utilize and integrate an API/library/technology that detects or help determines a homograph attack.
- Utilize and integrate an OCR and DIP API/library/technology that corrects or reduces image noise for more explicit images and for more accurate URL text extraction to create a correct input, resulting in a higher classification accuracy rate.
- Utilize and integrate a URL detector and extractor API/library with high URL extraction accuracy.
- Utilize and integrate multiple or powerful short URL full link extractor/ unshortener API/library to enable the mobile application to detect and unshortened shortened URLs from any different URL shortener services.
- Utilize cross-platform frameworks for mobile application development.

REFERENCES

[1] "Hackers Attack Every 39 Seconds." https://www.securitymagazine.com/articles/87787-hackers-attack-every-39-seconds (accessed Jan. 09, 2022).

[2] "Cost of a Data Breach Report 2020," p. 82.

[3] "Cyber Crime & Security | Statista." https://www.statista.com/markets/424/topic/1065/cyber-crime-security/#statistic1 (accessed Dec. 12, 2021).

[4] "Economic Impact of Cybercrime | Center for Strategic and International Studies." https://www.csis.org/analysis/economic-impact-cybercrime (accessed Dec. 13, 2021).

[5] "Cybercrime To Cost The World $10.5 Trillion Annually By 2025," *Cybercrime Magazine*, Dec. 08, 2018. https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/ (accessed Jan. 09, 2022).

[6] "Cybersecurity threats to cost organizations in the Philippines US$3.5 billion in economic losses – Microsoft News Center Philippines." https://news.microsoft.com/en-ph/2018/06/01/cybersecurity-threats-to-cost-organizations-in-the-philippines-us3-5-billion-in-economic-losses/ (accessed Dec. 13, 2021).

[7] "philippines GDP in 2018 - Google Search." https://www.google.com/search?q=philippines+GDP+in+2018&rlz=1C1CHBF_enPH981PH981&oq=philippines+GDP+in+2018&aqs=chrome..69i57j0i512l5j69i60l2.5941j0j9&sourceid=chrome&ie=UTF-8 (accessed Jan. 03, 2022).

[8] "What is the Cost of a Data Breach in 2021? | UpGuard." https://www.upguard.com/blog/cost-of-data-breach (accessed Jan. 10, 2022).

[9] "annual-report-2021.pdf." Accessed: Dec. 13, 2021. [Online]. Available: https://www.cityoflondon.police.uk/SysSiteAssets/media/downloads/city-of-london/about-us/annual-report-2021.pdf

[10] "Cybersecurity for SMBs: Asia Pacific Businesses Prepare for Digital Defense," p. 22, 2021.

[11] "Social Networking Service (SNS) Definition." https://www.investopedia.com/terms/s/social-networking-service-sns.asp (accessed Dec. 12, 2021).

[12] “Social Networking Sites - an overview | ScienceDirect Topics.” https://www.sciencedirect.com/topics/social-sciences/social-networking-sites (accessed Sep. 20, 2021).

[13] “Social Media Threats: Facebook Malware, Twitter Phishing, and More,” *Calyptix Security*, Aug. 17, 2017. https://www.calyptix.com/top-threats/social-media-threats-facebook-malware-twitter-phishing/ (accessed Sep. 20, 2021).

[14] “Phishers’ Favorites Top 25 H1 2021, Worldwide Edition.” https://www.vadesecure.com/en/blog/phishers-favorites-top-25-h1-2021-worldwide-edition (accessed Sep. 21, 2021).

[15] “Rising Threats of Social Media Phishing Attacks,” *Corrata*, Oct. 11, 2021. https://corrata.com/rising-threats-of-social-media-phishing-attacks/ (accessed Jan. 07, 2022).

[16] “Remove Facebook virus (Removal Instructions) - 2021 update.” https://www.2-spyware.com/remove-facebook-virus.html (accessed Sep. 21, 2021).

[17] “Social Media Platforms Double as Major Malware Distribution Centers.” https://www.darkreading.com/vulnerabilities-threats/social-media-platforms-double-as-major-malware-distribution-centers (accessed Sep. 22, 2021).

[18] N. Gupta, A. Aggarwal, and P. Kumaraguru, “bit.ly/malicious: Deep dive into short URL based e-crime detection,” in *2014 APWG Symposium on Electronic Crime Research (eCrime)*, Birmingham, AL, USA, Sep. 2014, pp. 14–24. doi: 10.1109/ECRIME.2014.6963161.

[19] “Most Abused TLDs.” http://www.surbl.org/tld (accessed Sep. 22, 2021).

[20] “Social Media Scams: 29 Disturbing Scam Statistics - InfoSec Insights.” https://sectigostore.com/blog/social-media-scams-29-disturbing-statistics/ (accessed Sep. 21, 2021).

[21] D. J. Guan, C.-M. Chen, Q.-K. Su, and T.-Y. Wang, “Malicious URL Detection on Facebook,” p. 17, 2011.

[22] P. Dewan and P. Kumaraguru, “Facebook Inspector (FbI): Towards automatic real-time detection of malicious content on Facebook,” *Soc. Netw. Anal. Min.*, vol. 7, no. 1, p. 15, Dec. 2017, doi: 10.1007/s13278-017-0434-5.

[23] “Facebook spammers make $200m just posting links, researchers say | Facebook | The Guardian.” https://www.theguardian.com/technology/2013/aug/28/facebook-spam-202-million-italian-research (accessed Dec. 12, 2021).

[24] “Working to Stop Misinformation and False News.” https://www.facebook.com/formedia/blog/working-to-stop-misinformation-and-false-news (accessed Dec. 12, 2021).

[25] S. Sinha, M. Bailey, and F. Jahanian, “Shades of grey: On the effectiveness of reputation-based “blacklists”,” in *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, Alexandria, VA, USA, Oct. 2008, pp. 57–64. doi: 10.1109/MALWARE.2008.4690858.

[26] “• Phishing: most targeted industries 2021 | Statista.” https://www.statista.com/statistics/266161/websites-most-affected-by-phishing/ (accessed Dec. 12, 2021).

[27] E. Buber, O. Demir, and O. K. Sahingoz, “Feature selections for the machine learning based detection of phishing websites,” in *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, Malatya, Sep. 2017, pp. 1–5. doi: 10.1109/IDAP.2017.8090317.

[28] “XGBoost Algorithm: Long May She Reign! | by Vishal Morde | Towards Data Science.” https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d (accessed Nov. 13, 2021).

[29] “Remove WhatsApp virus (Removal Instructions) - 2021 update.” https://www.2-spyware.com/remove-whatsapp-virus.html (accessed Jan. 07, 2022).

[30] “Remove Instagram virus (Removal Instructions) - 2021 update.” https://www.2-spyware.com/remove-instagram-virus.html (accessed Jan. 07, 2022).

[31] “UnionBank Customers Offered Php 10,000 in SMS Phishing Scam,” *UNBOX PH*, Feb. 15, 2022. https://unbox.ph/editorials/unionbank-customers-offered-php-10000-in-sms-phishing-scam/ (accessed Mar. 12, 2022).

[32] “URLs - Website Research - Research Guides at Central Michigan University Libraries.” https://libguides.cmich.edu/web_research/urls (accessed Oct. 29, 2021).

[33] “apwg_trends_report_q1_2021.pdf.” Accessed: Oct. 29, 2021. [Online]. Available: https://docs.apwg.org/reports/apwg_trends_report_q1_2021.pdf

[34] “Malicious links | Cyber.gov.au.” https://www.cyber.gov.au/acsc/view-all-content/glossary/malicious-links (accessed Sep. 20, 2021).

[35] G. Kandolkar and S. Usgaonkar, “Comparative Analysis of Various Machine Learning Techniques for Detecting Malicious Webpages,” *Int. J. Eng. Res.*, vol. 10, no. 06, p. 5.

[36] “IRJET-V6I65420190819-80896-40px67-with-cover-page-v2.pdf.”

[37] “What are the types of machine learning? | by Hunter Heidenreich | Towards Data Science.” https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f (accessed Oct. 29, 2021).

[38] “What is Supervised Learning? | IBM.” https://www.ibm.com/cloud/learn/supervised-learning (accessed Dec. 30, 2021).

[39] “Reinforcement Learning - an overview | ScienceDirect Topics.” https://www.sciencedirect.com/topics/neuroscience/reinforcement-learning (accessed Oct. 29, 2021).

[40] Shantanu, B. Janet, and R. Joshua Arul Kumar, “Malicious URL Detection: A Comparative Study,” in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, Coimbatore, India, Mar. 2021, pp. 1147–1151. doi: 10.1109/ICAIS50930.2021.9396014.

[41] F. Vanhoenshoven, G. Napoles, R. Falcon, K. Vanhoof, and M. Koppen, “Detecting Malicious URLs using Machine Learning Techniques,” p. 8.

[42] R. Patgiri, H. Katari, R. Kumar, and D. Sharma, “Empirical Study on Malicious URL Detection Using Machine Learning,” in *Distributed Computing and Internet Technology*, vol. 11319, G. Fahrnberger, S. Gopinathan, and L. Parida, Eds. Cham: Springer International Publishing, 2019, pp. 380–388. doi: 10.1007/978-3-030-05366-6_31.

[43] F. O. Catak, K. Şahinbaş, and V. Dortkardes, “(PDF) Malicious URL Detection Using Machine Learning,” *ResearchGate*. https://www.researchgate.net/publication/345763969_Malicious_URL_Detection_Using_Machine_Learning (accessed Oct. 29, 2021).

[44] A. Saleem Raja, R. Vinodini, and A. Kavitha, “Lexical features based malicious URL detection using machine learning techniques,” *Mater. Today Proc.*, p. S2214785321028947, Apr. 2021, doi: 10.1016/j.matpr.2021.04.041.

[45] S. R. Bingi, “Improving the classification rate for detecting Malicious URL using Ensemble Learning Methods,” p. 27.

[46] “Light GBM vs XGBOOST: Which algorithm takes the crown,” *Analytics Vidhya*, Jun. 12, 2017. https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/ (accessed Jun. 11, 2022).

[47] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[48] A. Nagpal, “Decision Tree Ensembles- Bagging and Boosting,” *Medium*, Oct. 18, 2017. https://towardsdatascience.com/decision-tree-ensembles-bagging-and-boosting-266a8ba60fd9 (accessed Jun. 11, 2022).

[49] “Ensemble Methods,” *Corporate Finance Institute*. https://corporatefinanceinstitute.com/resources/knowledge/other/ensemble-methods/ (accessed Jun. 11, 2022).

[50] “Fig. 2. eXtreme Gradient Boosting (XGBoost) Schematic Representation:...,” *ResearchGate*. https://www.researchgate.net/figure/eXtreme-Gradient-Boosting-XGBoost-Schematic-Representation-it-builds-decision-trees_fig2_357741497 (accessed May 20, 2022).

[51] “eXtreme Gradient Boosting.” Distributed (Deep) Machine Learning Community, Jun. 10, 2022. Accessed: Jun. 10, 2022. [Online]. Available: https://github.com/dmlc/xgboost

[52] D. Sahoo, C. Liu, and S. C. H. Hoi, “Malicious URL Detection using Machine Learning: A Survey,” *ArXiv170107179 Cs*, Aug. 2019, Accessed: Sep. 11, 2021. [Online]. Available: http://arxiv.org/abs/1701.07179

[53] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. Riley, “Are these Ads Safe: Detecting Hidden Attacks through the Mobile App-Web Interfaces,” San Diego, CA, 2016. doi: 10.14722/ndss.2016.23234.

[54] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, “Detecting Malicious URLs Using Lexical Analysis,” in *Network and System Security*, vol. 9955, J. Chen, V. Piuri, C. Su, and M. Yung, Eds. Cham: Springer International Publishing, 2016, pp. 467–482. doi: 10.1007/978-3-319-46298-1_30.

[55] “How it works – VirusTotal.” https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works (accessed Feb. 26, 2022).

[56] C. Amrutkar, Y. S. Kim, and P. Traynor, “Detecting Mobile Malicious Webpages in Real Time,” p. 14.

[57] “Lists.” http://www.surbl.org/lists (accessed Mar. 03, 2022).

[58] C. D. Xuan, H. Dinh, and T. Victor, “Malicious URL Detection based on Machine Learning,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 1, 2020, doi: 10.14569/IJACSA.2020.0110119.

[59] M. Darling, G. Heileman, G. Gressel, A. Ashok, and P. Poornachandran, “A Lexical Approach for Classifying Malicious URLs,” p. 8, 2015.

[60] H. Choi, B. B. Zhu, and H. Lee, “Detecting Malicious Web Links and Identifying Their Attack Types,” p. 12.

[61] A. B.Sayamber and A. M. Dixit, “Malicious URL Detection and Identification,” *Int. J. Comput. Appl.*, vol. 99, no. 17, pp. 17–23, Aug. 2014, doi: 10.5120/17464-8247.

[62] “OpenPhish - FAQ.” https://openphish.com/faq.html (accessed Feb. 28, 2022).

[63] “Christian Urcuqui | Contributor.” https://www.kaggle.com/xwolf12/competitions (accessed Mar. 03, 2022).

[64] “Malicious_n_Non-Malicious URL.” https://kaggle.com/antonyj453/urldataset (accessed Mar. 08, 2022).

[65] “1. Introduction,” *WOT Help Center*. https://support.mywot.com/hc/en-us/articles/360024397433-1-Introduction (accessed Mar. 07, 2022).

[66] “URL 2016 | Datasets | Research | Canadian Institute for Cybersecurity | UNB.” https://www.unb.ca/cic/datasets/url-2016.html (accessed Mar. 08, 2022).

[67] “About DMOZ.” https://dmoz-odp.org/docs/en/about.html (accessed Mar. 09, 2022).

[68] “URLhaus | Malware URL exchange.” https://urlhaus.abuse.ch/ (accessed Mar. 11, 2022).

[69] “Malicious URLs dataset.” https://www.kaggle.com/sid321axn/malicious-urls-dataset (accessed May 18, 2022).

[70] “SMS Spam Collection Dataset.” https://www.kaggle.com/uciml/sms-spam-collection-dataset (accessed May 18, 2022).

[71] “How to create a confusion matrix in Python using scikit-learn,” *Educative: Interactive Courses for Software Developers*. https://www.educative.io/edpresso/how-to-create-a-confusion-matrix-in-python-using-scikit-learn (accessed May 22, 2022).

[72] R. M, “Churning the Confusion out of the Confusion Matrix,” *Medium*, Sep. 24, 2019. https://blog.clairvoyantsoft.com/churning-the-confusion-out-of-the-confusion-matrix-b74fb806e66 (accessed May 22, 2022).

[73] “systemusabilityscale28sus29_comp5B15D.pdf.”

[74] “The System Usability Scale & How it’s Used in UX | Adobe XD Ideas,” *Ideas*. https://xd.adobe.com/ideas/process/user-testing/sus-system-usability-scale-ux/ (accessed May 27, 2022).

[75] A. Bangor, “Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale,” vol. 4, no. 3, p. 10, 2009.

[76] “Optical Character Recognition (OCR) - Overview and Use Cases,” *viso.ai*, Jun. 26, 2021. https://viso.ai/computer-vision/optical-character-recognition-ocr/ (accessed Nov. 10, 2021).

[77] F. Xue, B. B. Zhu, and W. Chu, “MALICIOUS UNIFORM RESOURCE LOCATOR DETECTION.” Oct. 02, 2014. [Online]. Available: https://patents.google.com/patent/US20140298460A1/en?oq=US20140298460

[78] “Check if a Website is Malicious/Scam or Safe/Legit | URLVoid,” *URLVoid.com*. https://www.urlvoid.com/ (accessed Nov. 09, 2021).

[79] “(1) New Messages!” https://www.lucidchart.com/blog/rapid-application-development-methodology (accessed Dec. 16, 2021).

[80] “Rapid Application Development (RAD) | Definition, Steps & Full Guide.” https://kissflow.com/low-code/rad/rapid-application-development/ (accessed Dec. 16, 2021).

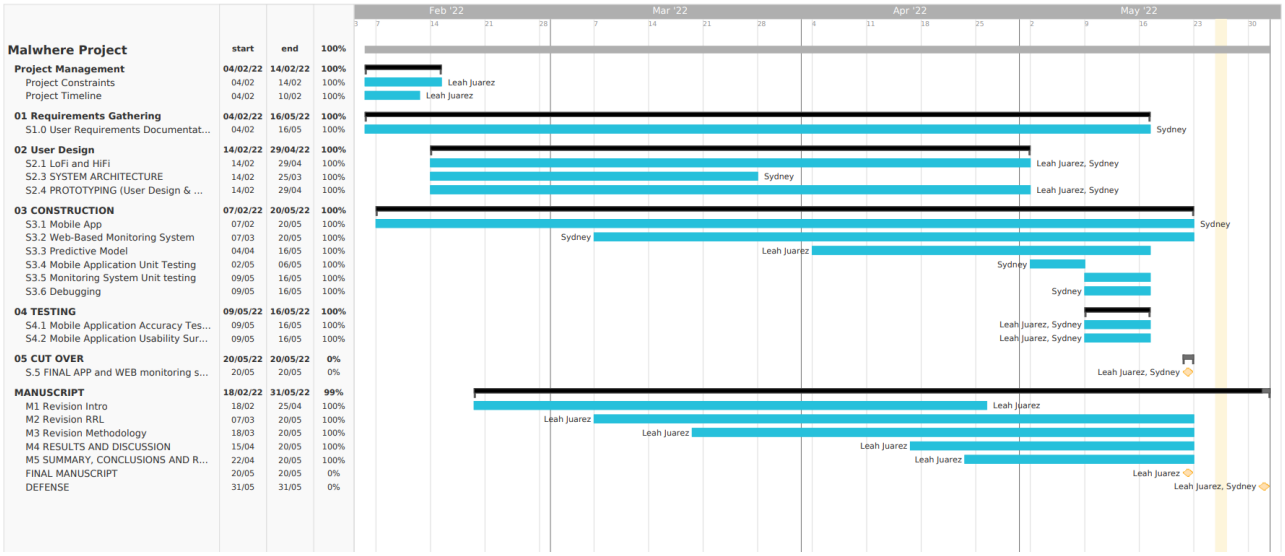
[81] “Recognize Text in Images with ML Kit on Android | Firebase Documentation.” https://firebase.google.com/docs/ml-kit/android/recognize-text (accessed Dec. 14, 2021).

[82] “Correlational Research | When & How to Use,” *Scribbr*, Jul. 07, 2021. https://www.scribbr.com/methodology/correlational-research/ (accessed May 26, 2022).

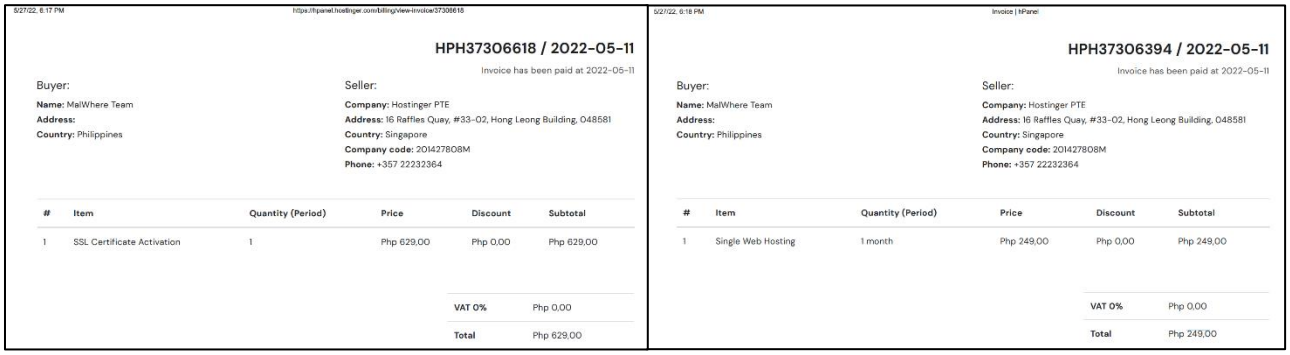
[83] “What is Flask Python - Python Tutorial.” https://pythonbasics.org/what-is-flask-python/ (accessed May 26, 2022).

APPENDIX A
REQUIREMENTS GATHERING PHASE RELEVANT SCREENSHOTS

PROJECT TIMELINE



INVOICES



5/27/22, 6:17 PM https://hpanel.hostinger.com/billing/view-invoice/37306618

HPH37306618 / 2022-05-11

Invoice has been paid at 2022-05-11

Buyer:
Name: MalWhere Team
Address:
Country: Philippines

Seller:
Company: Hostinger PTE
Address: 16 Raffles Quay, #33-02, Hong Leong Building, 048581
Country: Singapore
Company code: 201427808M
Phone: +357 22232364

#	Item	Quantity (Period)	Price	Discount	Subtotal
1	SSL Certificate Activation	1	Php 629,00	Php 0,00	Php 629,00

VAT 0%	Php 0,00
Total	Php 629,00

5/27/22, 6:18 PM Invoice | HPanel

HPH37306394 / 2022-05-11

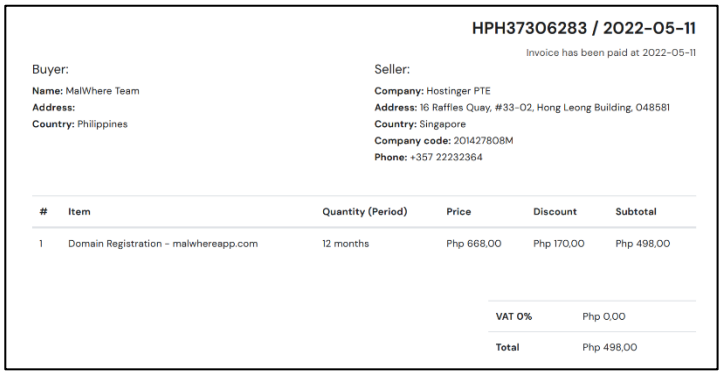
Invoice has been paid at 2022-05-11

Buyer:
Name: MalWhere Team
Address:
Country: Philippines

Seller:
Company: Hostinger PTE
Address: 16 Raffles Quay, #33-02, Hong Leong Building, 048581
Country: Singapore
Company code: 201427808M
Phone: +357 22232364

#	Item	Quantity (Period)	Price	Discount	Subtotal
1	Single Web Hosting	1 month	Php 249,00	Php 0,00	Php 249,00

VAT 0%	Php 0,00
Total	Php 249,00



HPH37306283 / 2022-05-11

Invoice has been paid at 2022-05-11

Buyer:
Name: MalWhere Team
Address:
Country: Philippines

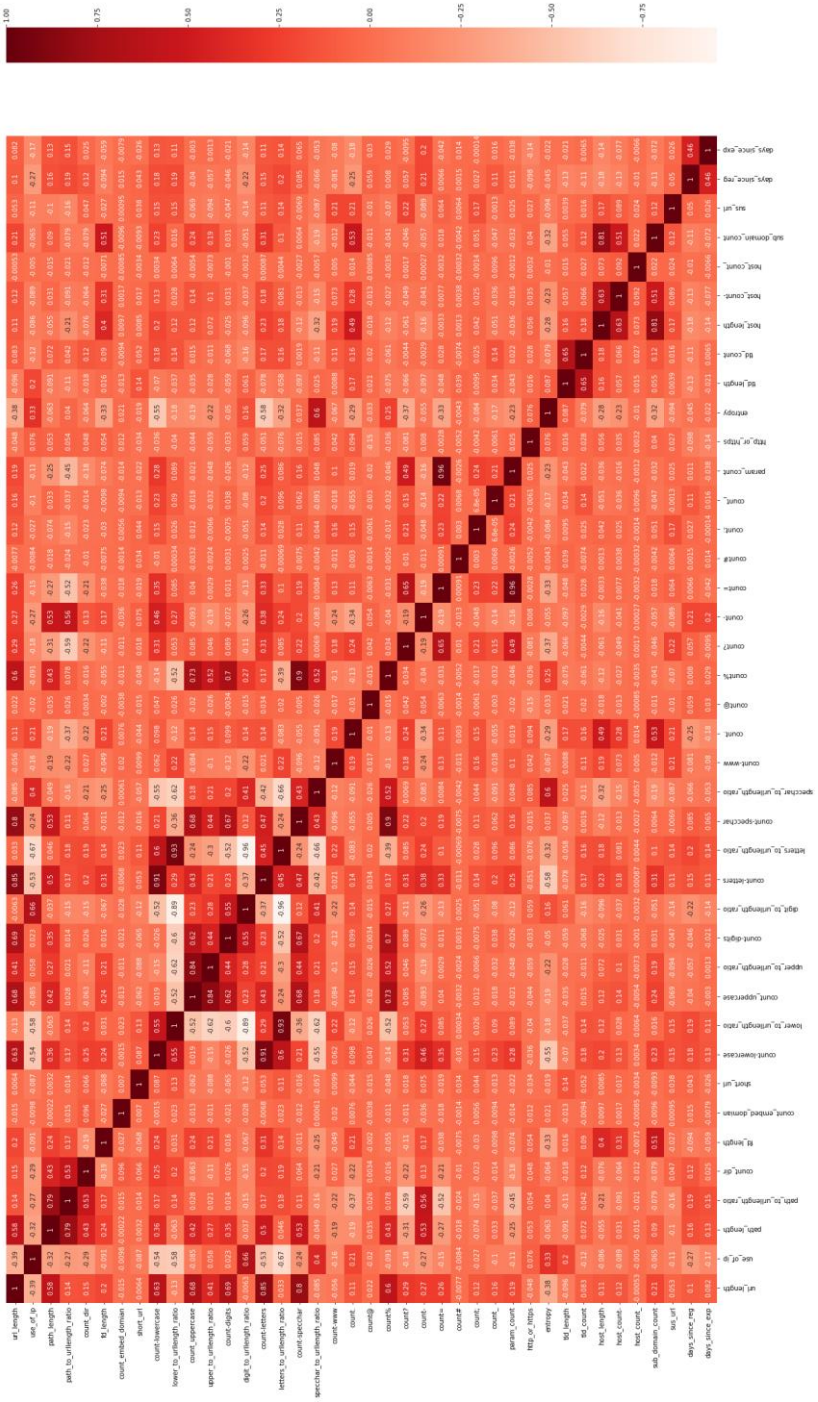
Seller:
Company: Hostinger PTE
Address: 16 Raffles Quay, #33-02, Hong Leong Building, 048581
Country: Singapore
Company code: 201427808M
Phone: +357 22232364

#	Item	Quantity (Period)	Price	Discount	Subtotal
1	Domain Registration - malwhereapp.com	12 months	Php 668,00	Php 170,00	Php 498,00

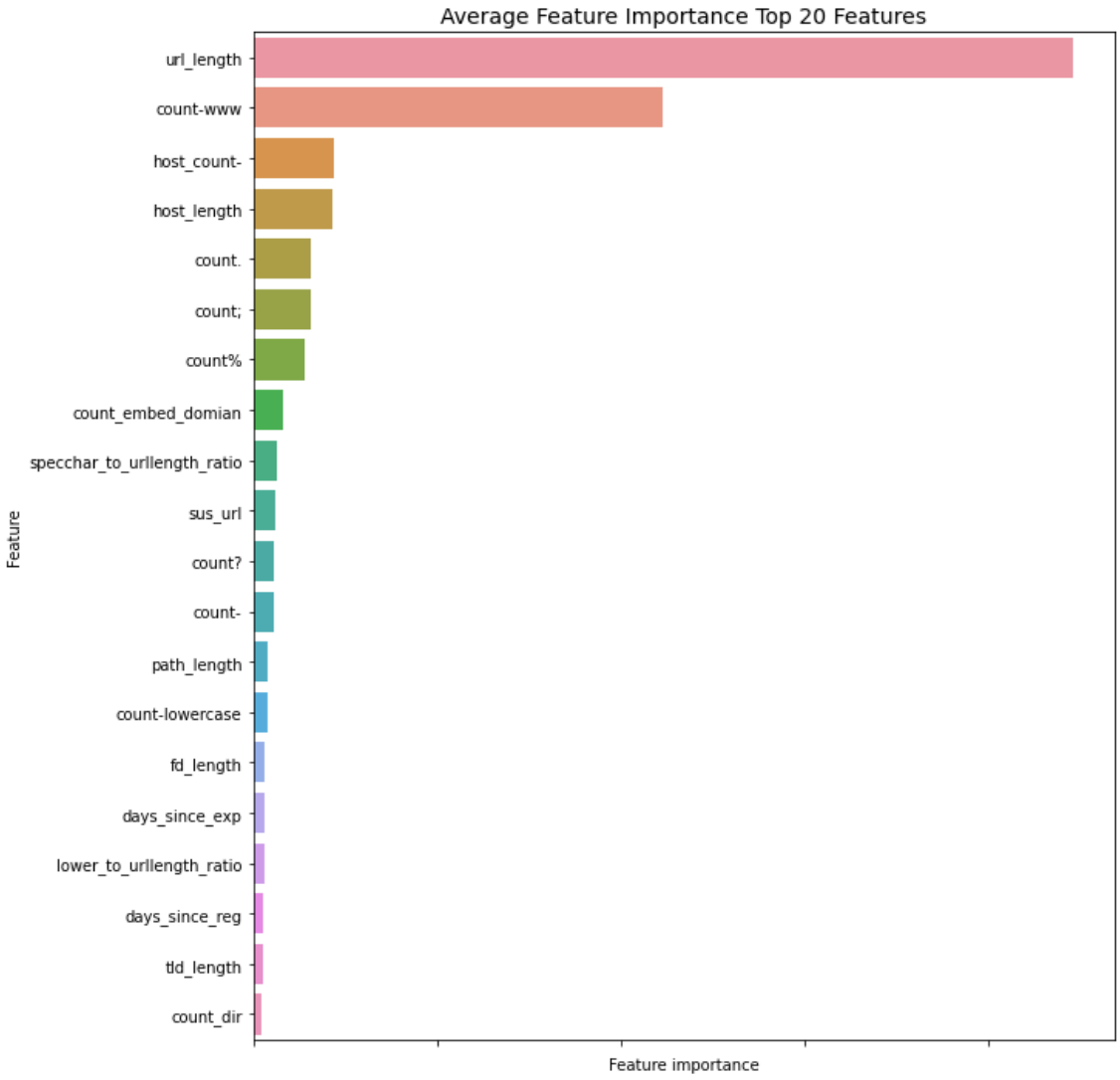
VAT 0%	Php 0,00
Total	Php 498,00

APPENDIX B
USER DESIGN PHASE RELEVANT SCREENSHOTS

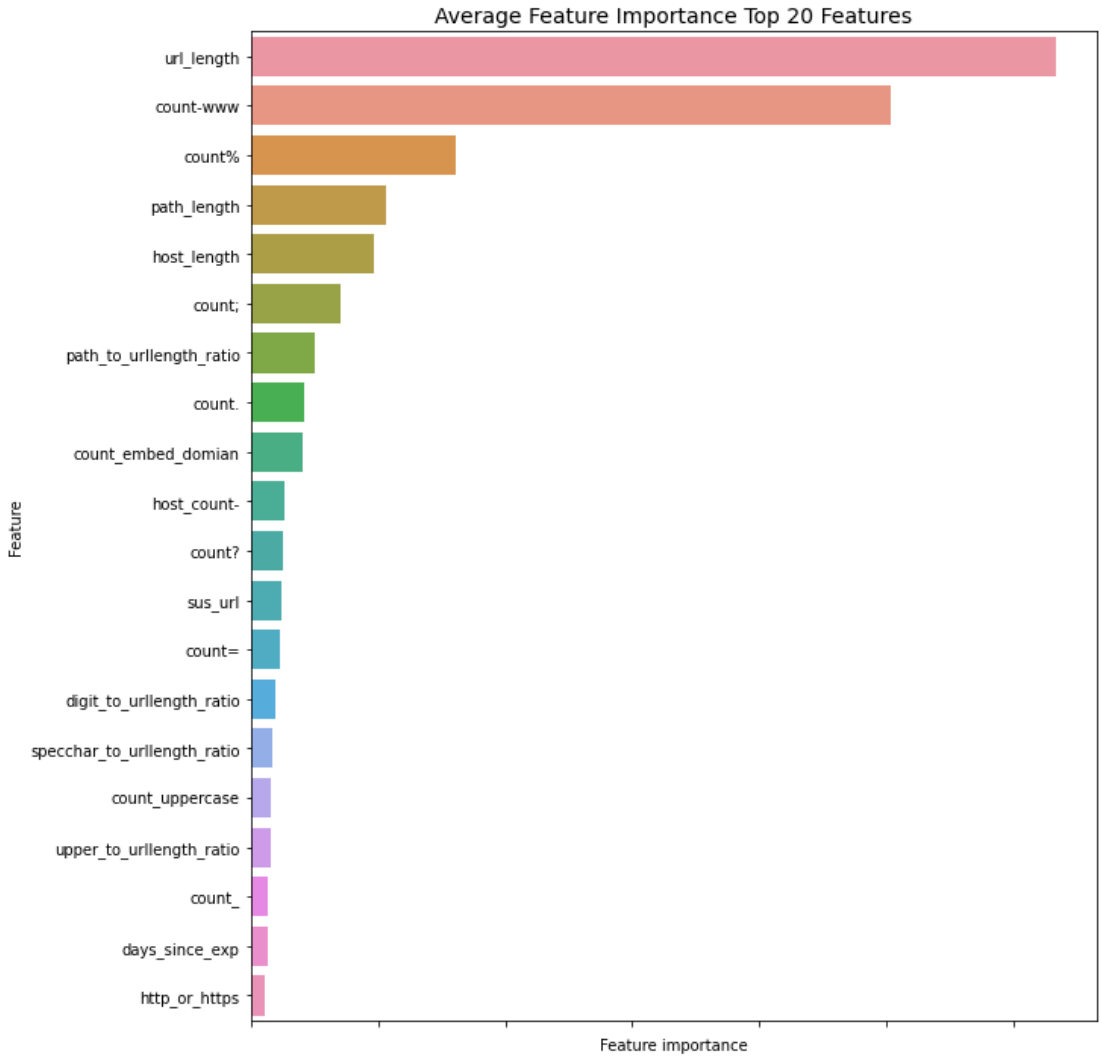
FEATURE CORRELATION OF THE INITIAL SELECTED FEATURES USED FOR FEATURE SELECTION



TOP 20 IMPORTANT FEATURES USED IN CREATING IN THE 1ST CLASSIFICATION MODEL



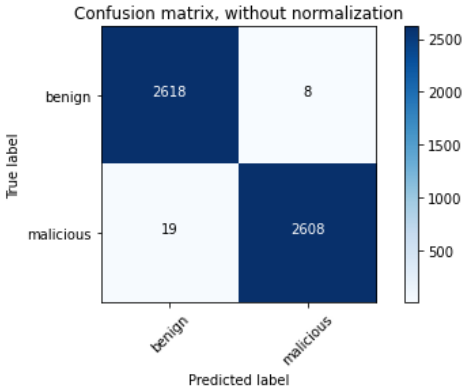
TOP 20 IMPORTANT FEATURES USED IN CREATING 2ND CLASSIFICATION MODEL



CONFUSION MATRIX OF THE 1st MODEL (with code snippet)

testing data = 5253

malicious= 2627; benign =2626

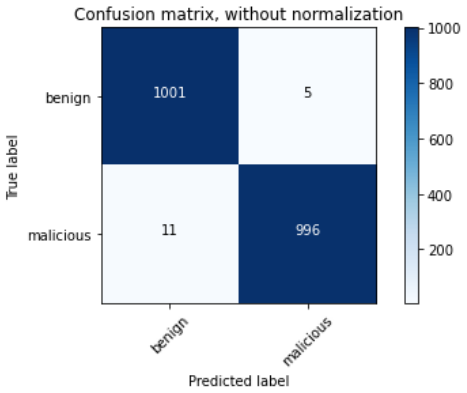



```
from plot_confusion_matrix import plot_confusion_matrix
cm = metrics.confusion_matrix(y_test, y_pred, labels=[0,1])
print (cm)
plot_confusion_matrix(cm,classes=['benign', 'malicious'] ,title='Confusion matrix, without normalization')
plot_confusion_matrix(cm,classes=['benign', 'malicious'],normalize="True",title='Normalized confusion matrix')

import itertools
import matplotlib.pyplot as plt
import numpy as np

#Create confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    See full source and example:
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
#       cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)
        cm= np.around(cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

CONFUSION MATRIX OF THE 2nd MODEL(with code snippet)

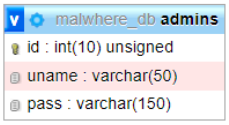
testing data = 2013

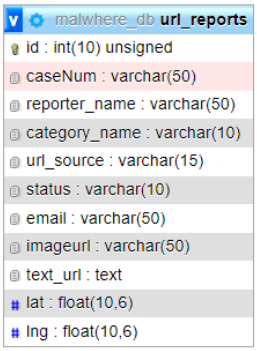
malicious= 1007; benign =1006

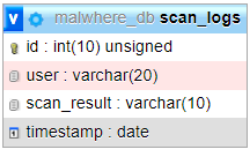



```
from plot_confusion_matrix import plot_confusion_matrix
cm2 = metrics.confusion_matrix(y_test2_s, y_pred2_s, labels=[0,1])
print (cm2)
plot_confusion_matrix(cm2,classes=['benign', 'malicious'] ,title='Confusion matrix, without normalization')
plot_confusion_matrix(cm2,classes=['benign', 'malicious'],normalize="True",title='Normalized confusion matrix')

import itertools
import matplotlib.pyplot as plt
import numpy as np

#Create confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    See full source and example:
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
#       cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)
        cm= np.around(cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

ERD OF WEB-BASED MONITORING SYSTEM (UNRELATIONAL TABLE)







DATA DICTIONARY – WEB MONITORING SYSTEM

malwhere_db

admins

Column	Type	Null	Default	Links to	Comments	Media type
id *(Primary)*	int(10)	No				
uname	varchar(50)	No				
pass	varchar(150)	No				

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	0	A	No	

scan_logs

Column	Type	Null	Default	Links to	Comments	Media type
id *(Primary)*	int(10)	No				
user	varchar(20)	No				
scan_result	varchar(10)	No				
timestamp	date	No				

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	21	A	No	

url_reports

Column	Type	Null	Default	Links to	Comments	Media type
id *(Primary)*	int(10)	No				
caseNum	varchar(50)	No				
reporter_name	varchar(50)	No				
category_name	varchar(10)	No				
url_source	varchar(15)	Yes	*NULL*			
status	varchar(10)	No				
email	varchar(50)	No				
imageurl	varchar(50)	No				
text_url	text	No				
lat	float(10,6)	No				
lng	float(10,6)	No				

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	2	A	No	

APPENDIX C
RELEVANT SOURCE CODE

MOBILE APP

EXTRACT BLOCK OF TEXT FROM AN IMAGE

```
//text recognition process
private void detectText() throws IOException {
    FirebaseVisionImage firebaseVisionImage = FirebaseVisionImage.fromFilePath( context: MainActivity.this,imageUri);
    FirebaseVisionTextRecognizer firebaseVisionTextRecognizer = FirebaseVision.getInstance()
            .getOnDeviceTextRecognizer();

    Task<FirebaseVisionText> result =
            firebaseVisionTextRecognizer.processImage(firebaseVisionImage)
                    .addOnSuccessListener(new OnSuccessListener<FirebaseVisionText>() {
                        @Override
                        public void onSuccess(FirebaseVisionText firebaseVisionText) {
                            // Task completed successfully
                            // ...
                            processText(firebaseVisionText);
                        }
                    })
                    .addOnFailureListener(
                            new OnFailureListener() {
                            @Override
                            public void onFailure(@NonNull Exception e) {
                                // Task failed with an exception
                                // ...
                            }
                    });
}

 private void processText(FirebaseVisionText text){
     List<FirebaseVisionText.TextBlock> blocks = text.getTextBlocks();

     if(blocks.size() ==0){
         Snacky.builder()
                 .setText("No Text")
                 .setIcon(R.drawable.ic_error_outline_black_24dp)
                 .info()
                 .show();

         return;
     }

     recognizedText = new StringBuilder();

     for (FirebaseVisionText.TextBlock block: text.getTextBlocks()){
          recognizedText.append(block.getText());
     }
     recogTextDialog();

 }
```

GENERATE SCAN REPORT FROM VIRUS TOTAL

```
public void getUrlReport(String myUrl){

    try {
        VirusTotalConfig.getConfigInstance().setVirusTotalAPIKey("0332bfc39c9ccbed656e5c29f2821e957b2c6a589c2f00887dc788e...");
        VirustotalPublicV2 virusTotalRef = new VirustotalPublicV2Impl();

        String urls[] = {myUrl};
        FileScanReport[] reports = virusTotalRef.getUrlScanReport(urls, scan: false);

        String temp = "";

        for (FileScanReport report : reports) {
            if(report.getResponseCode()==0){
                continue;
            }

            sbResult = new StringBuilder();

            Map<String, VirusScanInfo> scans = report.getScans();

            SpannableStringBuilder spannableStringBuilder = new SpannableStringBuilder();

            for (String key : scans.keySet()) {
                VirusScanInfo virusInfo = scans.get(key);

                if(virusInfo.getResult().contains("clean")){
                    cleanSiteCount++;
                }else if(virusInfo.getResult().contains("unrated")){
                    unratedSiteCount++;
                }else{
                    maliciousCount++;
                }

                sbResult.append(key + "......... " + virusInfo.getResult()).append("\n");

            }
```

EXTRACT LONG URL FROM SHORT URL

```
private void extractLongUrl(String urlShorten){

    RequestQueue queue = Volley.newRequestQueue( context: this);
    String myReq = "https://unshort.herokuapp.com/api/?url=" + urlShorten;

    // prepare the Request
    JsonObjectRequest getRequest = new JsonObjectRequest(Request.Method.GET, myReq, jsonRequest: null,
            new Response.Listener<JSONObject>()
            {
                @Override
                public void onResponse(JSONObject response) {
                    // display response
                    try {
                        //assign the long url to setter and getter
                        new Adapter().setDetected_URL(response.getString( name: "longUrl"));
                        //open new activity scan class
                        startActivity(new Intent( packageContext: MainActivity.this,ScanUrl.class));
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            },
            new Response.ErrorListener()
            {
                @Override
                public void onErrorResponse(VolleyError error) {
                    Log.d( tag: "Error.Response", error.toString());
                }
            }
    );

    // add it to the RequestQueue
    queue.add(getRequest);
}
```

CLASSIFICATION MODEL IMPLEMENTATION INTO MOBILE APPLICATION

```
private void predictiveModedl(String mUrl){
    RequestQueue queue = Volley.newRequestQueue( context: this);
    String myReq = "https://malwhere-model-api4.herokuapp.com/?url=" + mUrl;

    // prepare the Request
    JsonObjectRequest getRequest = new JsonObjectRequest(Request.Method.GET, myReq,  jsonRequest: null,
            new Response.Listener<JSONObject>()
            {
                @Override
                public void onResponse(JSONObject response) {
                    // display response
                    //assign the long url to setter and getter
                    try {

                        //Toast.makeText(ScanTextUrl.this,response.getString("prediction"),Toast.LENGTH_SHORT).show();
                        if(response.getString( name: "prediction").equalsIgnoreCase( anotherString: "[0]")){
                            BenignDialog benignDialog = new BenignDialog();
                            benignDialog.showDialog( activity: ScanUrl.this, msg: "Benign URL",  msg2: "No vendors flagged this URL as malicious");

                            //save log to server
                            String user = "";
                            String scanResult = "Benign";
                            String timeStamp = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(Calendar.getInstance().getTime());


                            if(sharedPreferences.getString( s: "guest_user", s1: "").isEmpty()){
                                user = sharedPreferences.getString( s: "user_display_name", s1: "");
                            }else{
                                user = sharedPreferences.getString( s: "guest_user", s1: "");
                            }

                            sbResult.append("\n").append("MalWhere Predictive Model: BENIGN");

                            tvScanResult.setText(sbResult.toString());

                            saveLog(user,scanResult,timeStamp);
                        }else{

                            MaliciousDialog maliciousDialog = new MaliciousDialog();
                            maliciousDialog.showDialog( activity: ScanUrl.this, msg: "Malicious URL",
                                    count: "This URL is flagged as Malicious");

                            //save log to server
                            String user = "";
                            String scanResult = "Malicious";
                            String timeStamp = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(Calendar.getInstance().getTime());

                            if(sharedPreferences.getString( s: "guest_user", s1: "").isEmpty()){
                                user = sharedPreferences.getString( s: "user_display_name", s1: "");
                            }else{
                                user = sharedPreferences.getString( s: "guest_user", s1: "");
                            }
                            sbResult.append("\n").append("MalWhere Predictive Model: MALICIOUS");

                            tvScanResult.setText(sbResult.toString());


                            saveLog(user,scanResult,timeStamp);

                        }

                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            },
            new Response.ErrorListener()
            {
```

VIRUS TOTAL IMPLEMENTATION- URL CLASSIFICATION

```
public void scanUrl(String myUrl) {
    try {
        VirusTotalConfig.getConfigInstance().setVirusTotalAPIKey("0332bfc39c9ccbed656e5c29f2821e957b2c6a589c2f00887dc788e...");
        VirustotalPublicV2 virusTotalRef = new VirustotalPublicV2Impl();

        String urls[] = {myUrl};
        ScanInfo[] scanInfoArr = virusTotalRef.scanUrls(urls);

        for (ScanInfo scanInformation : scanInfoArr) {
            Log.d(TAG, msg: "___SCAN INFORMATION___");

        }


    } catch (APIKeyNotFoundException ex) {
        Toast.makeText( context: this, text: "API Key not found! " + ex.getMessage(),Toast.LENGTH_SHORT).show();
    } catch (UnsupportedEncodingException ex) {
        Toast.makeText( context: this, text: "Unsupported Encoding Format!" + ex.getMessage(),Toast.LENGTH_SHORT).show();
    } catch (UnauthorizedAccessException ex) {
        Toast.makeText( context: this, text: "Invalid API Key " + ex.getMessage(),Toast.LENGTH_SHORT).show();
    } catch (Exception ex) {
        Toast.makeText( context: this, text: "Something Bad Happened! " + ex.getMessage(),Toast.LENGTH_SHORT).show();
    }
}
```

URL CLASSIFICATION-COMBINATION OF VIRUS TOTAL AND CLASSIFICATION MODEL

```
        if(maliciousCount > 0){

            MaliciousDialog maliciousDialog = new MaliciousDialog();
            maliciousDialog.showDialog( activity: ScanUrl.this, msg: "Malicious URL",
                    count: Integer.toString(maliciousCount) + "/" + Integer.toString(scans.size()) + " vendors flagged this URL as malicious");

            //save log to server
            String user = "";
            String scanResult = "Malicious";
            String timeStamp = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(Calendar.getInstance().getTime());

            if(sharedPreferences.getString( s: "guest_user", s1: "").isEmpty()){
                user = sharedPreferences.getString( s: "user_display_name", s1: "");
            }else{
                user = sharedPreferences.getString( s: "guest_user", s1: "");
            }

            tvScanResult.setText(sbResult.toString());

            saveLog(user,scanResult,timeStamp);

        }else{
            //if url is not blacklisted call predict model
            new PredictUrlModelTask().execute();
        }

    }

} catch (APIKeyNotFoundException ex) {
    Toast.makeText( context: this, text: "API Key not found! " + ex.getMessage(),Toast.LENGTH_SHORT).show();
} catch (UnsupportedEncodingException ex) {
    Toast.makeText( context: this, text: "Unsupported Encoding Format!" + ex.getMessage(),Toast.LENGTH_SHORT).show();

} catch (UnauthorizedAccessException ex) {
    Toast.makeText( context: this, text: "Invalid API Key " + ex.getMessage(),Toast.LENGTH_SHORT).show();
```

RELEVANT SOURCE CODE-WEB MONITORING SYSTEM

GEOTAGGING IMAGE

```
var map = new mapboxgl.Map({
  container: 'map',
  style: 'mapbox://styles/mapbox/dark-v10',
  center: [121.7740, 12.8797],
  zoom: 5
});

var geojson = <?php echo json_encode($geojson,JSON_NUMERIC_CHECK); ?>;

  // add markers to map
  geojson.features.forEach(function(marker) {

    // create a HTML element for each feature
    var el = document.createElement('div');
    el.className = 'marker';

    var finalImageUrl =marker.properties.imageurl.replace(/\\/g, '');


    // make a marker for each feature and add to the map
    new mapboxgl.Marker(el)
      .setLngLat(marker.geometry.coordinates)
      .setPopup(new mapboxgl.Popup({ offset: 25 }) // add popups
      .setHTML('<strong>Reporter: </strong>' + marker.properties.reporter_name + '<br>' + '<strong>Ca
      + marker.properties.status + "<br>" +  '<strong>Source: </strong>' + marker.properties.url_sour
      .addTo(map);
      });
</script>
```

REPORTED URL- URL CLASSIFICATION CONFIRMATION REPORT EMAIL MESSAGE

```
<?php
session_start();
require_once '../db_config.php';
require_once('../vendor/autoload.php');
ini_set("SMTP","files.000webhost.com");
ini_set("smtp_port","21");

use Nette\Mail\Message;
use Nette\Mail\SendmailMailer;

$mail = new Message;

$conn = new mysqli(DB_HOST,DB_USER,DB_PASS,DB_NAME);


//get vals
$reportID = mysqli_real_escape_string($conn,$_POST['reportID']);
$reporter = mysqli_real_escape_string($conn,$_POST['reporter']);
$url = mysqli_real_escape_string($conn,$_POST['url']);
$caseNum = mysqli_real_escape_string($conn,$_POST['caseNum']);
$category = mysqli_real_escape_string($conn,$_POST['category']);
$email = mysqli_real_escape_string($conn,$_POST['email']);

$messageBody  = "<p>Report ID: $caseNum</p><p>Reported URL: $url</p><p>Classification: Malicious </p>
<p>The reported url is 'Malicious'. We appreciate you for letting us know about this url.This informa
<p>Til your next report</p><br><p>-MalWhere Team</p>
<p style='font-style:italic'>Note: Do not reply to this email. This is a system generated email</p>";
```

CLASSIFY REPORTED URL MANUALLY

```
$("#approveBtn").click(function(){
    Swal.fire({
    title: 'Confirm Benign URL',
    text: "Are you sure you want to perform this action?",
    icon: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#FF5252',
    cancelButtonColor: '#CFD4D7',
    confirmButtonText: 'Approve'
  }).then((result) => {
    if (result.isConfirmed) {

     $.ajax({
        type:"POST",
        url:"../ajax_calls/benign_url.php",
        data:{reportID:reportID,email:email,url:url,caseNum:caseNum,category:category},
        dataType:"text",
        success:function(){

          Swal.fire({
              title: 'Success',
              text: " Operation was successful",
              icon: 'success',
              confirmButtonColor: '#FF5252',
              confirmButtonText: 'Ok'
              }).then((result) => {
              if (result.value) {
              location.reload();
              }
```

VIEW REPORTED URLS

```
    <?php foreach($reports as $report) : ?>
      <tr>
        <td style="display:none"><?= $report['id'];?></td>
        <td><?= $report['caseNum']; ?></td>
        <td><?= $report['category_name']; ?></td>
        <td><?= $report['url_source']; ?></td>
        <td><span id="<?= $report['status'] ?>"><?= $report['status'] ?></span></td>
        <td style="display:none"><?= $report['email']; ?></td>
        <td>
        <a href="#" class="pop">
        <img id="img" data-toggle="tooltip" title="View image" class="border " width="60px" src="<
        </a>
        </td>
        <td style="display:none"><?= $report['text_url']; ?></td>
        </tr>
        <?php endforeach; ?>
      </tbody>
    </table>
</div>
</div><br>
<form method="post"><button data-toggle="tooltip" title="Download report" class="btn btn-secondary fl
    Download</button></form>
```

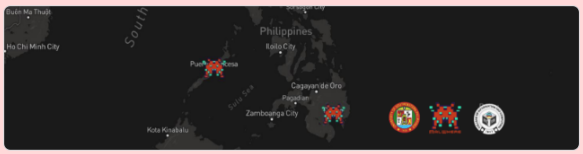
VIEW DASHBOARD

```
<script>
// Set new default font family and font color to mimic Bootstrap's default styling
// Area Chart Example
var ctx = document.getElementById("myAreaChart");
var myLineChart = new Chart(ctx, {
  type: 'line',
  data: {
    labels: [<?php while ($row = mysqli_fetch_array($resultArea1)) { echo '"' . $row['category_name']
    datasets: [{
      label: "Reports",
      lineTension: 0.3,
      backgroundColor: "rgba(2,117,216,0.2)",
      borderColor: "rgba(2,117,216,1)",
      pointRadius: 5,
      pointBackgroundColor: "rgba(2,117,216,1)",
      pointBorderColor: "rgba(255,255,255,0.8)",
      pointHoverRadius: 5,
      pointHoverBackgroundColor: "rgba(2,117,216,1)",
      pointHitRadius: 50,
      pointBorderWidth: 2,
      data:[<?php while ($row = mysqli_fetch_array($resultArea2)) { echo '"' . $row['COUNT(*)'] . '",
    }],
  },
  options: {
    scales: {
      xAxes: [{
        time: {
          unit: 'date'
```

```
<script>
    var ctx = document.getElementById("myPieChart1");
    var myPieChart = new Chart(ctx, {
      type: 'pie',
    data: {
      labels:  [<?php while ($row = mysqli_fetch_array($resultPie1)) { echo '"' . $row['scan_result']
      datasets: [{
        data: [<?php while ($row = mysqli_fetch_array($resultPie2)) { echo '"' . $row['COUNT(*)'] . '
        backgroundColor: ['#008140','#AB0501' ],
      }],
    },
  });
```

```
<script>
    var ctx = document.getElementById("myPieChart1");
    var myPieChart = new Chart(ctx, {
      type: 'pie',
    data: {
      labels:  [<?php while ($row = mysqli_fetch_array($resultPie1)) { echo '"' . $row['scan_result']
      datasets: [{
        data: [<?php while ($row = mysqli_fetch_array($resultPie2)) { echo '"' . $row['COUNT(*)'] . '
        backgroundColor: ['#008140','#AB0501' ],
      }],
    },
  });
  </script>
```

APPENDIX D
FEEDBACK AND USABILITY SURVEY



MalWhere App User Feedback Survey

In this study, you will be answering questions related to the application that we developed as a requirement to our Capstone Project, entitled, "DESIGN AND DEVELOPMENT OF A MOBILE-BASED MALICIOUS URL DETECTION APPLICATION".

By clicking the "Next" button below, you are voluntarily participating and giving permission to the researchers to use the information you provide for the attainment of the objectives of the project.

We ensure that your information will be kept confidential in compliance with Republic Act 10173 or the Data Privacy Act of 2012, as well as the Data Privacy Policy of USeP. Participation in this study is voluntary. You can also choose to leave the study at any time but, the data collected (up to the point of our withdrawal) will remain in our database.

lcjuarez@usep.edu.ph Switch account

* Required

Email *

lcjuarez@usep.edu.ph

Next Page 1 of 3 Clear form

Never submit passwords through Google Forms.

This form was created inside of University of Southeastern Philippines. Report Abuse

MalWhere App

A malicious link is an infected URL that facilitates scams, frauds, and a cyber-attacks. By clicking on a malicious URL, you can download a malware such as a Trojan or virus that can take control of your devices, stole your social media account or tricked you into disclosing your bank details on a fake website. To assist internet users on being vigilant on clicking malicious URLs, the researchers developed MalWhere. MalWhere is a mobile application that will analyze and classify if a URL is malicious (dangerous) or benign (safe). In comparison with blacklist lookup, MalWhere does not only classify recorded/known distributed malicious URLs, but also predicts and classify new or unknown distributed URLs.

Example of a URL:
https://www.google.com/
https://www.lawinsider.com/dictionary/messaging-platform

(MalWhere is a capstone project, developed in fulfillment of a requirement for our bachelor's degree in Information Technology)

As part of our research, we the researchers needs to know your criticisms/opinion/feedbacks as an internet user.
Please help us improve and measure the performance of MalWhere by using the app and by answering the questions honestly. (This survey will only take 10-35 minutes)

IMPORTANT NOTE! Please read:
This app is currently compatible to ANDROID phones with VERSIONS 6-10 and is 10.85mb.

To know which Android version you have:
1. Open your phone's Settings app.
2. Tap About phone
3. See your Android version

You can download the application using this link :

Link 1: https://tinyurl.com/2p82x2c6

Link 2: https://tinyurl.com/yswzcarf

Thank you and God bless.
For concerns and inquiries, please message us on: malwhereteam@gmail.com

What android version are you using? *

- ○ version 6
- ○ version 7
- ○ version 8
- ○ version 9
- ○ version 10
- ○ Other...

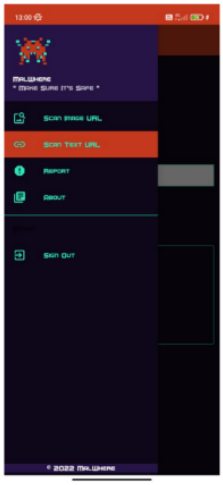
Have you already encountered malicious URLs before? How many times? *

- ○ Never
- ○ Once
- ○ Twice
- ○ More than twice

There are actually 3 methods to let the mobile app get the URL:(1) camera ;(2) screenshot; and (3) copy and paste. Which method does not function properly? If there is, please describe the anomaly. *

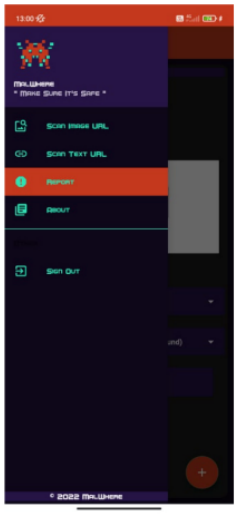
Long answer text

Scan text URL is a feature located at the app menu. Which of its function does not work properly? If there is, please describe the anomaly. *



Long answer text

There is a feature that can let you report a malicious URL on the menu. Which of its function does not work properly? If there is, please describe the anomaly. *



Long answer text

Is the app results accurate to you? How many times does the application labeled the suspicious URLs as "malicious"? *

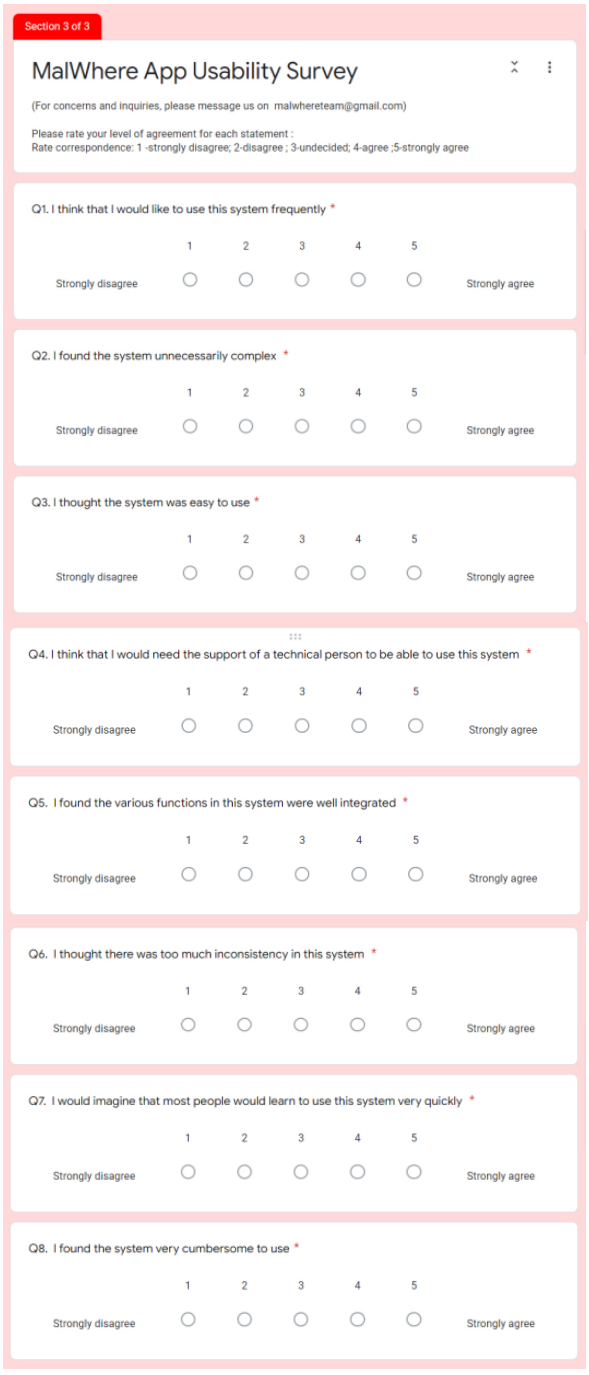
- Never
- once over 3 attempts (1/3)
- once over 4 attempts (1/4)
- twice over 4 attempts (2/4)
- thrice over 4 attempts (3/4)
- results malicious in all 4 attempts (4/4)
- Other:

Is the app results accurate to you? How many times does the application labeled the safe URLs as "benign" ? *

- Never
- once over 4 attempts (1/4)
- twice over 4 attempts (2/4)
- thrice over 4 attempts (3/4)
- accurate in all four attempts (4/4)
- Other...

How long did it take for the detection result to show? *

- 1 second
- 3 seconds
- 10 seconds
- Other:



Section 3 of 3

MalWhere App Usability Survey

(For concerns and inquiries, please message us on malwhereteam@gmail.com)

Please rate your level of agreement for each statement :
Rate correspondence: 1 -strongly disagree; 2-disagree ; 3-undecided; 4-agree ;5-strongly agree

Q1. I think that I would like to use this system frequently *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q2. I found the system unnecessarily complex *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q3. I thought the system was easy to use *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q4. I think that I would need the support of a technical person to be able to use this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q5. I found the various functions in this system were well integrated *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q6. I thought there was too much inconsistency in this system *

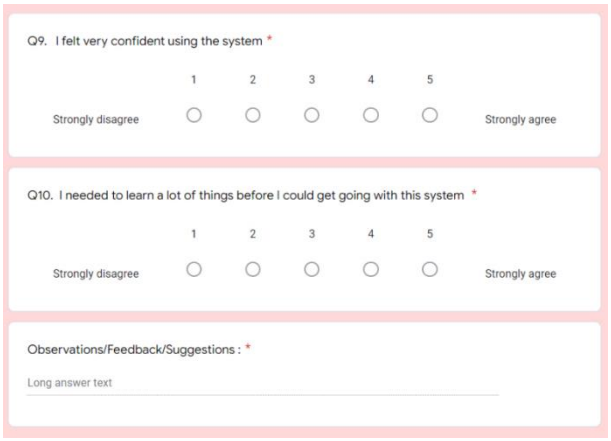
	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q7. I would imagine that most people would learn to use this system very quickly *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Q8. I found the system very cumbersome to use *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree


Q9. I felt very confident using the system *
1 2 3 4 5
Strongly disagree
Strongly agree
Q10. I needed to learn a lot of things before I could get going with this system *
1 2 3 4 5
Strongly disagree
Strongly agree
Observations/Feedback/Suggestions : *
Long answer text

APPENDIX E
UNIT TEST REPORT

Functions	Description	%TCs Executed	TCs Passed	TCs Pending	Priority	Remarks
Mobile Application						
Functional requirements						
FR1-Login flow	The mobile application shall allow the user to sign in using their Google account or sign in anonymously.	100%	2	0	High	Completed
FR2- Input image	The mobile app shall accept images in different formats such as jpeg, jpg and png.	100%	2	0	High	Completed
FR3- crop image	The system shall allow the user to crop image in different aspect ratio.	100%	2	0	High	Completed
FR4- Extract text from image	the mobile application shall extract text from image .	100%	2	0	High	Completed
FR5- Extract URL from a text	The mobile application shall be able to extract URL from recognized characters in	100%	2	0	High	Completed

	standard format. URL that are formatted in HTML 5 scheme , IPv4 address, IPv4 Decimal, IPv6 Address, IPv4-mapped IPv6 address are included.					
FR6- Extract the long URL of a shortened URL	the mobile application shall detect a short URL and extract its long URL.	100%	2	0	High	Completed
FR7- Copy & paste URL input method	the mobile application shall have an input text field as another option for URL input.	100%	2	0	High	Completed
FR8- URL classification	the mobile application shall classify the extracted final URL whether it is benign or malicious using the integrated and combined blocklist and machine learning classifiers.	100%	2	0	High	Completed
FR9- User URL Report	The mobile application shall allow	100%	2	0	High	Completed

	user to submit a report and require them to fill in required information before enabling them to submit a report.					
FR10- Scan result	The mobile application shall allow user to see the classification result from the blocklist service and the classification model(if the case is applicable).	100%	2	0	High	Completed
FR11- Scan meaning	The mobile application shall allow user to see the meaning of the classification result.	100%	2	0	High	Completed
FR12- Sign out	The mobile application shall allow user to sign out from the session.	100%	2	0	High	Completed
Non-Functional requirements						
NFR1- URL classification combination sequence.	the mobile app shall classify the final URL input using the integrated and combined blocklist and machine	100%	2	0	High	Completed

	learning classifiers. The combination shall operate in sequence. The primary URL classifier must be the blocklist service, given that it has large dataset of malicious URL . The secondary classifier must be the classification model. The model will be used when the URL blocklist will return a benign or unknown result. This technique is an attempt to lessen the false negative rate of a blocklist.					
NFR2-Security	- the mobile app must be implemented with google SSO to allow user to authenticate their account.	100%	2	0	High	Completed
NFR3-Usability	- The mobile application shall provide ease of use	100%	2	0	High	Completed

Web-based Monitoring System						
Functional requirements						
SFR1- View Dashboard	The system will enable the admin to view the summarized information of reported URL through graphical representation.	100%	2	0	High	Completed
SFR2-Manage reported URL	The system will enable the admin to view , confirm the URL classification , download and send automatic email response regarding the true classification of the reported URL	100%	2	0	High	Completed
SFR3- View Map	The system will enable the admin to view the location of reporters, using geotagging technology.	100%	2	0	High	Completed
FR4- Sign out	The system will enable user to revoke their active sessions.	100%	2	0	High	Completed
Non-Functional requirements						

SNFR1-Access security	The system will implement authentication using username and password to restrict database and information access from unauthorized person.	100%	2	0	High	Completed
SNFR2-Availability	The system shall be available any time of the day or throughout "normal operating times" to be able to perform its features and function.	100%	2	0	High	Completed
SNFR3-Usability	The system shall provide ease of use	100%	2	0	High	Completed
SNFR4-Confidentiality	The system shall protect all data and information and permits only authorized users to gain access on the data by disabling the directory listing.	100%	2	0	High	Completed

APPENDIX F
CURRICULUM VITAE

SYDNEY P. RICAFORT

EXPERIENCE

Software Developer

- Programmer on government high school management system on July 2021 to December 2021 under the supervision of USEP-CIC Extension Unit.
- Developer of USeP-Bantay Covid
- Developer of DA Survey Questionnaire System
- USeP- Design and Development of Mobile-based Malicious URL Detection Application (Capstone Project)
  February 2022 to May 2022

PROFESSIONAL SKILLS

Coding Languages:

C#, C++, JavaScript, HTML/CSS, Python, Visual Basic, Java

Frameworks/Systems:

Bootstrap, React-Bootstrap, jQuery, AJAX

EDUCATIONAL BACKGROUND

University of Southeastern Philippines Obrero Campus (College)

2018 - Present
Bachelor of Science in Information Technology

Holy Cross of Bunawan Inc. (Senior High School)

2016 - 2018

Accountancy and Business Management (ABM)

CERTIFICATION

Tesda Certification
NC 2 Computer Systems Servicing

Programming Competition, Pista ng mga Pirata
Chamption

ABOUT ME

I am a 4th year college student who is passionate in learning new and creating new things such as developing software that will help the community.

LANGUAGES

Filipino,Bisaya,English

MY CONTACT

Email:

spricafort@usep.edu.ph

Address:

Purok 14 Quitulao Bunawan Davao City

REFERENCE

Cheryl Amante
Unit Head, CIC Extension Unit
email: cramante@usep.edu.ph

Randy S. Gamboa
OJT Supervisor
email: rsgamboa@usep.edu.ph

Hermoso J. Tupas Jr
Capstone Project Adviser
email: hermoso.tupas@usep.edu.ph

LEAH C. JUAREZ

EXPERIENCE

Software Development & Project Management
University of Southeastern Philippines- (CIC Extension Unit)
July 2021 to December 2021

- Create and monitor tasks and timelines using a project management tool (Trello)
- Understand and design database
- Create documentation
- Develop and implement a high school record management system based on defined user requirements as part of a development team.

USeP- Design and Development of Mobile-based Malicious URL Detection Application (Capstone Project)
February 2022 to May 2022

- Create and monitor tasks and timelines using a project management tool
- Create a Research paper
- Develop a URL classification model API using a supervised machine learning technique

SKILLS

Programming Languages:
C++,Java, HTML/CSS, JavaScript, Python, PHP, SQL

Frameworks
Bootstrap, Ajax, Flask

Other Software Skills:
Photoshop, Adobe Illustrator, Adobe Animate, Figma, Filmora

EDUCATIONAL BACKGROUND

University of Southeastern Philippines (College)
2018 - 2022
BSIT-Major in Information Security

Holy Child College of Davao (Senior High School)
2016 - 2018
ICT - Specialized in Animation

CERTIFICATIONS

Block Chain
Block Chain Technology Workshop

Technopreneurship
Class Technopreneurship team pitching competition (2018)
3rd place

Space Apps Team Competition (2020)
3rd Place

ABOUT ME

I am a 4th-year college student with an interest in learning to develop web, desktop, and mobile applications to use as a solution to real world-problems or just simply for entertainment. Moreover, I have great interest in learning anything related to information security (e.g., networking, and infosec system researches)

LANGUAGES

English, Tagalog, Bisaya

MY CONTACT

Email:
lcjuarez@usep.edu.ph

Address:
Davao City

REFERENCE

Cheryl R. Amante
Unit Head, CIC Extension Unit
email: cramante@usep.edu.ph

Randy S. Gamboa
OJT Supervisor
email: rsgamboa@usep.edu.ph

Hermoso J. Tupas Jr
Capstone Project Adviser
email: hermoso.tupas@usep.edu.ph

APPENDIX E
GRAMMARIAN'S CERTIFICATE

G R A M M A R I A N' S C E R T I F I C A T E

This is to certify that the undersigned has reviewed and went through all the pages of the capstone project study manuscript entitled "**DESIGN AND DEVELOPMENT OF A MOBILE-BASED MALICIOUS URL DETECTION APPLICATION** " by Leah C. Juarez & Sydney P. Ricafort ,aligned with the set of structural rules that govern the composition of *sentences, phrases,* and *words* in the English language.

Signed:

Date Signed :

August 15, 2022

MR. ANGELO P. MARQUEZA
Signature over Printed Name

Grammarian