BSc Project

Computer Science

Project ID: UQU-CS-2022-08

November 2022

Developing a Robust Android System Towards Malicious URLs

Afrah Ibrahem Alsaadi	ID: 439008815
Afnan Musa Munshi	ID: 439004844
Jomanah Omar Bajahlan	ID: 439002377
Nedaa Atef Elgazzar	ID: 439014419
Lujain Samer Batouq	ID: 439006070

Dept. of Computer Science

Faculty of Computer and Information Systems

Umm Al-Qura University, KSA

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

﴿ وَأَنْ لَيْسَ لِلإِنسَانِ إِلَّا مَا سَعَى ﴾ [النجم:39]

This project report is submitted to the Department of Computer Science at Umm Al-Qura University in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Authors:
Afrah Alsaadi
Email: afrah.alsaadi.20@gmail.com

Afnan Munshi
Address: Al-Buhairat / Mecca
Email: fmmrm123@gmail.com

Jomanah Bajahlan
Address: Al-Khadraa / Mecca
Email: jomanahbajahlan@gmail.com

Nedaa Elgazzar
Address: Al-Naseem / Mecca
Email: nedaa.atef6@gmail.com

Lujain Batouq
Address: Waly Al-Ahad /Mecca
Email: Lujainsameer357@gmail.com

University supervisor:

Sarah Al-Shareef
Dept. of Computer Science			Internet: http://uqu.edu.sa
Faculty of Computer and Information Systems		Phone: +966 xxxxxxxxx
Umm Al Qura University			Fax : +966 xxxxxxxxx

Kingdom of Saudi Arabia

Intellectual Property Right Declaration

This is to declare that the work under the supervision of Sarah Al-Shareef having title "Developing a Robust Android System Towards Malicious URLs" carried out in partial fulfillment of the requirements of Bachelor of Science in Computer Science, is the sole property of the Umm Al Qura University and the respective supervisor and is protected under the intellectual property right laws and conventions. It can only be considered/ used for purposes like extension for further enhancement, product development, adoption for commercial/organizational usage, etc., with the permission of the University and respective supervisor.

This above statement applies to all students and faculty members.

Date: ______________

Authors:

Name: Lujain Samer Batouq	Signature: [illegible]
Name: Jomanah Omar Bajahlan	Signature: ج.ع.ب
Name: Nedaa Atef Al Gazzar	Signature: [illegible]
Name: Afnan Musa Munshi	Signature: [illegible]
Name: Afrah Ibrahem Alsaadi	Signature: [illegible]

Supervisor:

Name: Sarah Al-Shareef Signature: ___SAH_____________

Anti-Plagiarism Declaration

This is to declare that the above publication produced under the supervision of Sarah Al-Shareef having title “Developing a Robust Android System Towards Malicious URLs” is the sole contribution of the author(s) and no part hereof has been reproduced illegally (cut and paste) which can be considered as plagiarism. All referenced parts have been used to argue the idea and have been cited properly. I/We will be responsible and liable for any consequence if violation of this declaration is proven.

Date: ______________

Authors:

Name: Lujain Samer Batouq	Signature:	[illegible]
Name: Jomanah Omar Bajahlan	Signature:	[illegible]
Name:Nedaa Atef Al Gazzar	Signature:	[illegible]
Name: Afnan Musa Munshi	Signature:	[illegible]

Name: Afrah Ibrahem Alsaadi
Signature: [illegible]

ACKNOWLEDGMENTS

This work is dedicated to our dear parents, the most loving in this world.

ABSTRACT

Malware threats become more dangerous, and every application we use daily can get hacked in multiple ways. In recent years many attackers used malicious URLs to download viruses, trojans, ransomware, or any other type of malware that will compromise machines or networks. Hence, we propose a machine-learning-based solution to detect these URLs in the background without requiring additional steps from the user. First, we will train a machine learning (ML) model to classify URLs into malicious or safe ones using one or more classification algorithms. This machine-learning-based solution helps make analytics smarter and faster, with the ability to scale alongside ever-increasing amounts of data. Unlike most available tools, which deploy the model on a mobile application, we customized the Android OS to embed our malicious URL detector so all URLs in any application will be checked in the background once the user clicks on them from any installed application. The project employed Random Forest model with a competitive performance, then the model was converted and deployed to Java. After that, a version of the Android Open Source Project was downloaded, customized with the malicious URL detector, built and tested on the Emulator.

Keywords: URL, malicious URL detection, customized android system, lexical features, machine learning, cybersecurity.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abb.	Full term
AB	AdaBoost
AI	Artificial Intelligence
AOSP	Android open-source project
API	Application Programming Interface
AUROC	Area Underneath the Receiver Operating Characteristic Curve
BN	Bayer-Net
CPU	Central Processing Unit
CTI	Cyber Threat Intelligence
DS	Decision Stump
DT	Decision Tree
DVM	Dalvik Virtual machine
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FTP	File Transfer Protocol
GB	Gradient Boost
GBDT	Gradient Boosted Decision Trees
GNB	Gaussian Naive Bayes
HT	HoeffdingTree
IPC	Inter-Process Communications,
JVM	Java Virtual Machine
KNN	k-Nearest Neighbor
KTF	Keras-TensorFlow
LC	Lazy Classifier
LD	Linear Discriminant
LR	Logistic Regression
MCC	Matthews Correlation Coefficient

ML	Machine Learning
MuD	Malicious URL Detection
NB or NBC	Naïve Bayes Classifier
QD	Quadratic Discriminant
RAT	Remote Access Trojan
RF	Random Forest
RT	Rep Tree
SMTP	Single Mail Transfer Protocol
SVM	Support Vector Machine
TLD	Top-Level Domain
TP	True Positive
TN	True Negative
URL	Uniform Resource Locator

Chapter 1 – INTRODUCTION

Malware threats become more dangerous, and every application we use daily can be hacked. Hacking is intentionally accessing a system without authorization or exceeding authorized access. Hacking can include any unconventional way of interacting with systems, i.e., interaction in a way not foreseen as a standard by the system designer [1]. One of the most common attacks is phishing, a type of social engineering where an attacker sends a fraudulent, spoofed, fake, or otherwise deceptive message designed to trick a person into revealing sensitive information to the attacker [2]. Phishing attacks have increased recently, as they hit 46% more companies in 2021 [3].

1.1 Problem Statement

Most of the current virus scanners are URL scanners or website checkers, such as *Sucuri* and *PhishTank* [4]; however, none are embedded with the operating system to decide whether to access the URL or not. Such applications require users to pause their activities, copy the URL in question, and paste it into these applications. Moreover, these applications are based on a URL blacklist, which must be updated frequently.

1.2 Objectives

As a result, a solution is needed to protect the users from malicious URLs by analyzing and classifying any URL before accessing it. Such a solution should make its decisions smoothly in the background without interrupting users' activities unless blocking them from accessing a harmful URL. Moreover, this solution should be updated with the increased volume of the malicious URL list.

With the help of ML, a model can be trained to classify any URL as safe or malicious. Such a model will not need frequent updating as a stored list [5]. Then, we will modify an Android OS to be robust to malicious URLs by embedding the malicious URL detector within the system. Modifying the operating system will guarantee that every URL can be screened in the background, regardless of which application is requesting the URL.

1.3 Report Organization

This chapter introduced the problem statement and the objectives this project is trying to achieve. The rest of the report is organized as follows:

Chapter 2 gave a simple background about the architecture of the Android system, its main layers, and components, with a brief review of each component, how the kernel works, its security, and its limitations. It also reviews URLs and their components, what malicious URLs are, and their risks, and gives some tools available to detect them. Also, it includes background about how machine learning works for detecting malicious URLs and discusses the related work.

Chapter 3 introduces the system requirements, the customized system's solution, and other alternative solutions that can be applied.

Chapter 4 introduces the design constraints (hardware and software environments), and gives more details about the used algorithm, how we will reuse the existing software parts, the development methodology, and the future plans.

Chapter 5 details our ML modeling methodology. It also includes the dataset and tools that will be used.

Chapter 6 will discuss our android design and how its structure interacts and behaves when it detects malicious URLs.

Chapter 7 will cover all the steps of implementation and the results of the algorithms used in the ML model and compare them.

Chapter 8 will discuss enhancing these limitations and how it works only in web view, the size of the model in java, and our future plans.

Appendices include attempts to customize the AOSP, attempts of the build model, and code reference of the project in GitHub.

Chapter 2 – BACKGROUND AND RELATED WORK

This project aims to customize an Android operating system with an embedded machine-learning-based malicious URL detector. Hence, it is necessary to have a brief understanding of the Android system to be customized, how URLs work, and how machine learning was used to detect malicious ones. This chapter provides the necessary background on these topics.

2.1 Android Operating System

This section will cover the android system architecture and briefly review each component, how the kernel works, and its security and limitations.

2.1.1 Android Architecture

Android architecture has five major components. Figure1. illustrates the main architecture of the Android system and their relationship to each other.

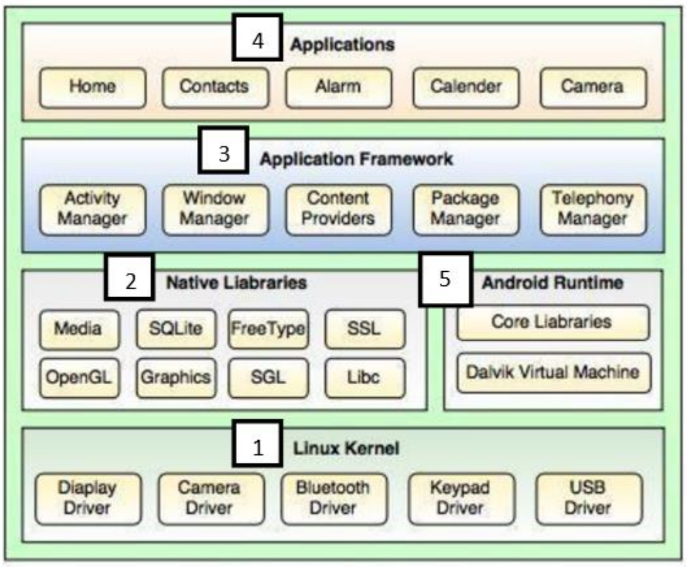


Figure 1. Android Operating system is a stack of software components that can be divided into five layers. 1) Kernel Layer. 2) Native Libraries layer. 3) Application Framework layer. 4) Application Layer.5) Android Runtime layer. Taken from [U Farooq - Lahore, 'Android Operating System Architecture', Virtual University of Pakistan, 2018.].

I. Linux Kernel

Android uses a version of the Linux kernel with additional special features. Some of these features include a low memory killer, a memory management system that is more aggressive in preserving memory; wake locks, a PowerManger system service; the binder IPC driver; and other essential features for a mobile embedded platform [12]. The main job of the kernel is to get the work done from the hardware. Here, the hardware can be anything in your device, for example, the camera, keyboard, or Wi-Fi. In other words, the kernel serves as an interface between the application and hardware [5].

The kernel has three primary processes: process management (CPU), memory management, and device management. Figure 2. depicts how the kernel serves as an interface between these processes and the applications.

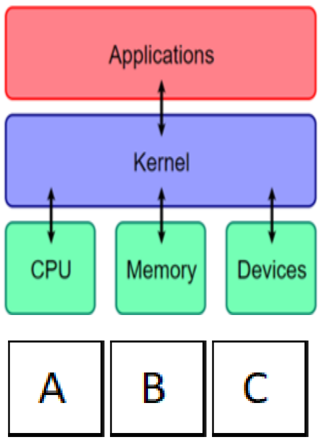


Figure 2. Android kernel has three processes: process management (CPU), memory management, and device management. Taken from [Josh (2015) Demystifying the Linux kernel, Digilent Blog. Available at: https://digilent.com/blog/demystifiying-the-linux-kernel/ (Accessed: October 1, 2022).]

A. *CPU Management.* A process is a series of steps and decisions involved in the way work is completed [6]. Therefore, everything a computer can perform is essentially a process, such as turning the Bluetooth on and off or taking pictures. CPU Management maintains all these processes.
B. *Memory Management.* Every process starts on the system demands recourses from the kernel (System Call). So, the kernel analyses the

process and allocates memory, and decides the amount of needed recourses for each one. So, it manages the entire memory.

C. *Device Drivers.* This process drives the work from hardware such as memory chips, Wi-Fi hardware, and camera drivers.

II. Native Libraries

Now, after the kernel, we have the libraries, which are some logical groups or some logical instructions that we want to give the kernel to perform an action, for instance, OpenGL libraries, C/C++ libraries, and java interfaces libraries.

III. Application Framework

This is the third layer on top of the kernel and the native libraries. It is basically an Application Programming Interface or API for short. APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols [7].

It has several groups of instances to help developers in programming their applications. These interfaces make application development easier and untestable. For example, the telephony manager helps to send SMS and make calls, so if your application needs to send SMS, you don’t have to go through a deeper layer; you can directly use this component.

IV. Applications

Here we have some applications already built on our phones, like contact and alarm. These applications, or apps for short, are available on the Android phone itself. Therefore, as a developer, you build an application above this layer.

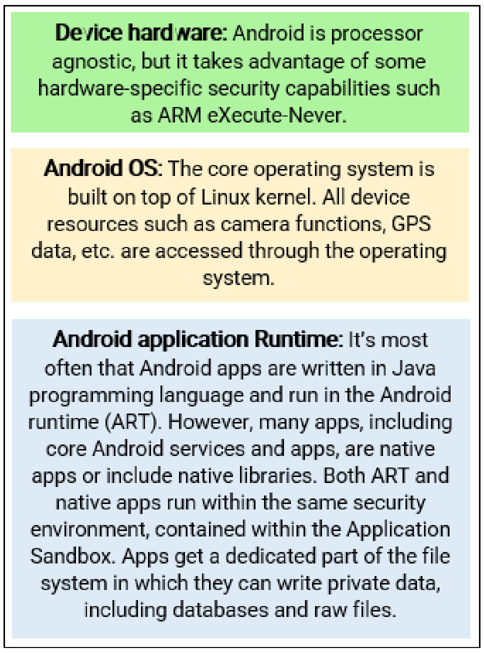
V. Android Runtime

It has a Dalvik Virtual machine (DVM) and core Java libraries. Android applications and the fundamental frameworks are written in Java programming language. Java has always been marketed as “write once, run anywhere”. The capability has largely been made possible by Java Virtual

Machine (JVM). Instead of using a standard Java Virtual Machine (JVM), Android uses its own Dalvik virtual machine. DVM provides portability and runtime consistency and enables each application to run with the required resources [8].

This component executes files in the Dalvik executable (.dex), which is more efficient than class files. Also, it reduces battery and memory usage.

With these five components, an Android-based device can be efficiently used. Figure 3. provides a summarized overview of the system.



Device hardware: Android is processor agnostic, but it takes advantage of some hardware-specific security capabilities such as ARM eXecute-Never.

Android OS: The core operating system is built on top of Linux kernel. All device resources such as camera functions, GPS data, etc. are accessed through the operating system.

Android application Runtime: It's most often that Android apps are written in Java programming language and run in the Android runtime (ART). However, many apps, including core Android services and apps, are native apps or include native libraries. Both ART and native apps run within the same security environment, contained within the Application Sandbox. Apps get a dedicated part of the file system in which they can write private data, including databases and raw files.

Figure 3. Main android platform building blocks

2.1.2 Android Security

According to [12], Android is an open-source platform; therefore, securing an open platform requires a strong security architecture and intensive security programs. Multi-layered security is designed within Android to support an open platform, protect all users, and provide an app environment that protects the confidentiality, integrity, and availability of users, data, apps, devices, and the network.

In addition, Android provides security controls to reduce the hard work on developers, which they can work easily with and rely on. Default security controls protect developers who are less familiar with security.

Android was designed to reduce the probability of attacks such as social engineering, which is “to convince users to install malware,” and attacks on third-party apps also highly limit the impact of a successful attack. Android works with partners and the public to provide patches for any android device still receiving security updates.

Each component in the Android software architecture assumes that the component below is properly secured. The application sandbox restricts all code above the Linux kernel. There are two levels of Android security: Kernel and program security.

I. *Kernel and Platform Security*

Since the Linux kernel is the foundation of the Android platform, the Linux kernel provides Android with several key security features like permissions model, process isolation, extensible mechanism for secure IPC, and the ability to remove unnecessary and potentially insecure and weak parts of the kernel [15].

The Android platform also provides key security features, such as:

- *Robust security at the OS level through the Linux kernel*
- *Apps sandbox*:

  According to [11], Android was created with certain security design principles, such as privilege separation. At the application level, as we mentioned previously, all code above the Linux kernel is restricted by the application sandbox, and that makes Android a widely deployed system that uses privilege separation as a matter of course. This sandboxing isolates apps from each other. This isolation prevents many types of unauthorized access and information exposure, e.g., application A accessing sensitive information stored on the system or in the private space of application B, performing unauthorized communications, or accessing features in the hardware such as the camera or GPS. Instead, applications employ unique permissions at the application level, like READ SMS, to ask for access to system resources, which the user must approve. The permissions paradigm on Android prevents unauthorized access by applications. It restricts access to features that either directly or indirectly cause financial harm by requiring each application to request permission to access protected resources expressly. Since the Application Sandbox is a part of the kernel, native code and OS apps are protected by this security architecture. Figure 4 views an example of the sandbox.

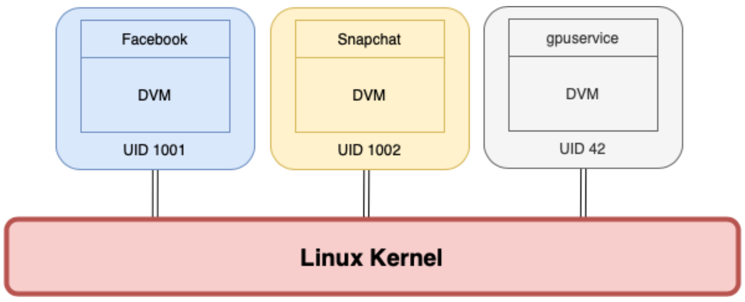


Figure 4. each application is sandboxed and isolated from the other by the kernel, taken from [13].

- *Secure inter-process communication:*

  One of the inter-process communications (IPC) mechanisms that Android employs is known as Intents which is a simple message object that stands for an "intention" to do something [12]. Intents are the ideal mechanism to transfer data when dealing with activities or broadcast listeners. Intents can be explicit or implicit.

  According to Crăciunescu [13], explicit intents are intended to be interpreted by an explicit component. In this manner, we may be certain that application B alone receives the data given by application A. Additionally, data can be sent across activities inside our applications using explicit intents.

  Implicit intents define an action that must be performed. Depending on the action, the intent may contain some required data for that particular action. Implicit intents are usually employed when our application cannot do an action and we want to assign the task to a third-party application.

- *App signing:*

  The developer must sign each Android application before it may be used. Google Play or the package installer on an Android device will refuse any application that try to install without signing [14].

- *App-defined and user-granted permissions:*

  The set of protected APIs on Android now includes APIs allowing user data access. Android smartphones will naturally collect user data within any third-party applications users install. Applications that desire to share this data can employ Android OS permission checks to safeguard the information from applications from other developers [12].

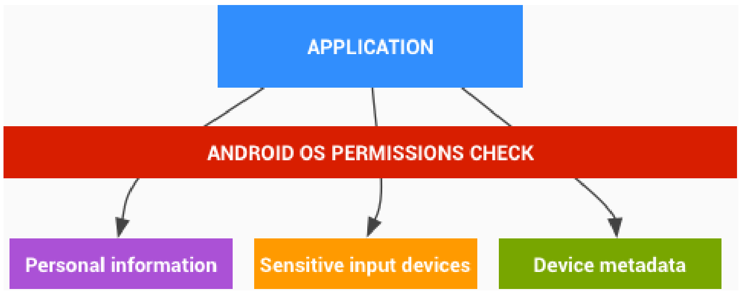


Figure 5. Access to sensitive user data is available only through protected APIs. Taken from [12]

II. *Application security*

The key components of the app's security include:

Design review: The Android security process begins early in the development lifecycle. Engineering and security resources review each feature, using appropriate security controls integrated into the system's architecture.

Penetration testing and code review: These reviews and tests carried out during the development process aim to identify weaknesses and possible vulnerabilities well before releases.

Open source and community review: Since Android is open-source, anyone interested can review the app.

Incident response: In the event of reporting certain bugs, Android has a comprehensive security response process that enables fast mitigation of threats.

Monthly security updates: These are updates provided by the Android security team.

2.1.3 Limitations

According to Umer Farooq [8], the security of Android OS has always been a concern compared to Apple’s iOS and BlackBarry OS, etc. The security of Android is considered low. Some of the well-known security risks are:

- Android is an open-source project; anyone can customize it. Whenever a bug, loophole, or weakness is reported through a public repository, android resolves that weakness and provides a patch to prevent attacks. Before the availability of security updates or in case of not deploying updates, there are chances of attacks.
- Components in Linux are interconnected and composed all in one piece; no isolation between them is provided. That makes Linux kernel inadequate for secure systems. When a kernel gets exploited, attackers modify the kernel memory, which can cause much harm.
- The device won’t be secure if the application or users root it since the rooting process overcomes the kernel integrity barrier.
- A major and dangerous security issue is giving irrelevant and/or unnecessary access to applications while installing apps.
- Platforms that are slowly patched during exhibits are regularly at greater risk due to the patches being available with compatible systems. These systems can be analyzed to determine the vulnerable conditions, making the unpatched devices an easy target due to the slow patches in Android. Once the software is reused, it will be more vulnerable [11]. Figure 6 illustrates the lifecycle of a security patch.

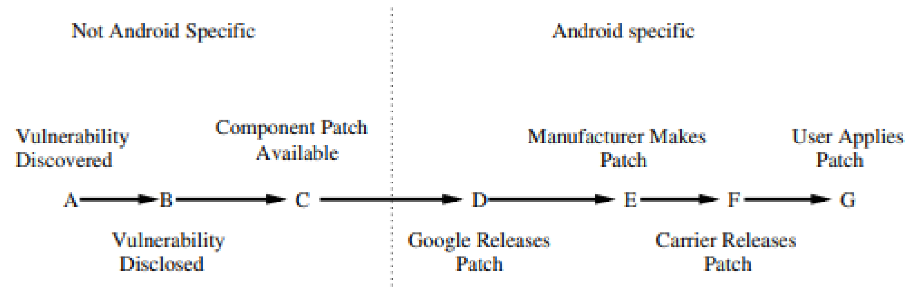


Figure 6. the cycle of Android patches from identifying vulnerability until the patch reaches the user’s device.

2.2 Malicious URL detection

This section briefly reviews URLs and their components. It also describes how a URL can be malicious and its risks. Finally, it gives some tools available to detect malicious URLs.

2.2.1 URLs Component

A URL (Uniform Resource Locator) is a special identifier used to locate resources on the Internet. Another name for it is a web address. A protocol and a domain name are just two of the numerous parts that make up a URL, which tells a web browser how and where to find a resource, as stated in [16].

According to [17], a URL comprises several components, some of which are required and others optional. A common website's URL contains at least three parts, such as www.facebook.com. Still, more complicated URLs may have up to nine components, including a scheme, subdomain, domain name, top-level domain, port number, path, query, parameters, and fragment, as depicted in Figure 7. In the following, each component will be described.

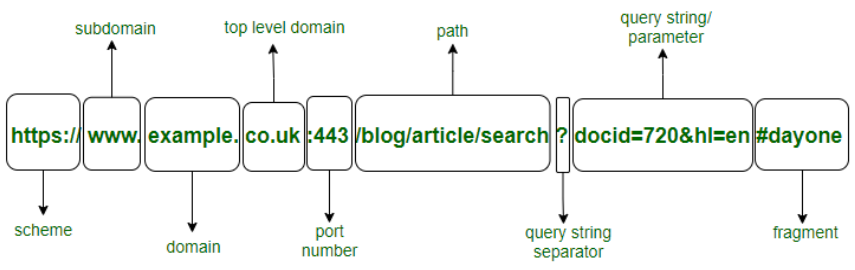


Figure 7. URL Components

I. Scheme

The protocol or scheme element of the URL denotes the set of rules that will govern data transport and exchange. HTTPS, which stands for Hyper Text Transfer Protocol Secure, instructs the browser to show the page in Hyper Text (HTML) format while also encrypting any information entered by the user. FTP, or File Transfer Protocol, is used for moving files between client and server, and SMTP, or Single Mail Transfer Protocol, is used for email transmission.

II. ***Subdomain***

The subdomain is used to distinguish between various portions of the website and specify the type of resource supplied to the client. The subdomain 'www' used here is a generic identifier for any online resource. Subdomains such as 'blog' point to a blog page, whereas 'audio' denotes that the resource type is audio.

III. ***Domain Name***

The domain name identifies the company or entity to which the URL belongs. The domain name 'facebook', as in www.facebook.com, denotes the organization that controls the site.

IV. ***Top-level Domain***

The TLD (top-level domain) specifies the sort of organization that owns the website. A business company is indicated by the.com in www.facebook.com. Similarly,.org denotes an organization, while.co.uk denotes a business firm in the United Kingdom.

V. ***Port Number***

Because servers sometimes provide different services, a port number identifies the type of service requested by the client. HTTP servers use port 80, whereas HTTPS servers use port 443.

VI. ***Path***

The path gives the precise location of the web page, file, or other resources to which the user wishes to gain access. For example, the path here denotes a specific article on the blog webpage.

VII. ***Query String Separator***

A question mark (?) precedes the query string, which includes the search parameters. The question mark indicates to the browser that a specific query is being run.

VIII. ***Query String***

The query string provides the parameters of the requested data from a website's database. Each query string consists of a parameter and a value separated by an equal (=) sign. When there are numerous arguments, query strings are concatenated together using the ampersand (&). The argument might be a number, a string, an encrypted value, or any other type of database data.

IX. ***Fragment***

A URL's fragment identifier is optional, occurs at the end, and begins with a hash (#). It denotes a specific place inside a page, like an HTML element's 'id' or 'name' property.

2.2.2 Malicious URLs and their risks

A malicious URL is a clickable link that takes people to a web page or website that is harmful or otherwise false. As its name indicates, a harmful URL can never result in anything positive. This is so because the motivation behind generating these malicious web pages is frequently evil, such as advancing a political agenda, stealing confidential information from businesses, or making fast money. Cybercriminals could produce malicious URLs, as reported by [20].

Because so many people click links without thinking, dangerous links are becoming more prevalent. When individuals receive a link, they just click it without giving it any thought.

According to[18], people who check their email on their phone or laptop while off-site are typically not protected by URL filtering and other network services; therefore, they usually get around most preventive measures. Since most people don't hover over links to check where they lead, it is simple for the bad actors to rename the links to make them appear different. Which could produce malicious URLs, for instance, to:

- Using phishing scams to get consumers' personal data to commit identity theft or fraud.
- Obtaining user login information to enter their personal or professional accounts.
- Exploiting deceptive tactics to get users to download malicious software that hackers can use to monitor victims' activities or take control of their devices.
- Entering the target's computer to encrypt the files in preparation for a ransomware attack.
- Using a remote access trojan to control a victim's computer from a distance (RAT). Additionally, an attacker can use RATs to build a botnet that they can command if they can spread them to other networked susceptible devices.

2.2.3 Malicious URLs detection tools

According to [19], falling into the trap of malicious URLs can be avoided through the following tips:

- If the URL is lengthy and the domain is unfamiliar to you, hover over it to reveal the link instead of clicking.
- Never open abbreviated URLs in emails. Because you can't hover over shortened URLs to see where they go, clicking links created using Bit.ly or other services poses a danger. They can be used to conceal a harmful website.
- Look at the email as a whole; was it what you were expecting? Do you know who sent it?
- Does the email appear to be focused on the link? If the email is only a greeting with a link, the likelihood that it is malicious is very high.
- Did you request it if the link was for a password change or something similar? To confirm its legitimacy, make direct contact with a reliable source.

And by using some tools that help to detect malicious URLs that [21] mentioned :

- *URL Void* :URL-checking program using multiple engines and blacklists of domains. Such as Google SafeBrowsing, Norton SafeWeb, and MyWOT.
- *UnMask Parasites:* which is a testing URLs tool.
- *Comodo Site Inspector:* This is a tool for detecting malware and security flaws. This allows users to examine URLs or webmasters to schedule daily checks by downloading all the selected sites. and execute them in a browser sandbox.
- *Antivirus software:* a program or product is made to find and get rid of viruses and other types of malicious software from your laptop or computer[22]. Like Sophos Intercept X for Mobile.[24]
- *Online resources to detect malicious URLs*: like VirusTotal[23] which is an online tool for investigating dubious URLs. The URLs linked to malware or other dangerous activities will be reported by this website, which makes use of various anti-malware engines.
- *Applications:* that detect URLs on mobile phones such as Link Protector: URL Security on Android[43].

And some other URL-checking tools, as reported by [21], such as UnShorten.it, Norton Safe Web, SiteAdvisor (by McAfee), Sucuri, Browser Defender, Online Link Scan, and Google Safe Browsing Diagnostic.

2.3 Machine learning for malicious URL detection

This project proposed a machine-learning-based malicious URL detector. Several algorithms are to be used for this purpose; therefore, a thorough review of the most recent research in detecting malicious URLs is required. This section includes background about how machine learning works for detecting malicious URLs. Also, it discusses the related work.

2.3.1 Supervised Learning Framework

The primary framework consists of two main stages: training and testing. Each stage is composed of several phases. The model's parameters are estimated using some training samples in the training stage. The trained model is then tested using some unseen samples in the testing stage.

First, data is collected from different sources, or an appropriate public dataset is chosen. After that, the collected data is pre-processed. Data pre-processing might include several steps, such as removing duplicates, handling missing data values, and data normalization. In the supervised learning paradigm, each sample is associated with an output. The clean data is split into three major subsets: training, validation, and testing, with the training set as the largest subset.

In the training stage, discriminative features are extracted from each sample. The training features with their corresponding output are used to estimate the parameter of some selected algorithms, while the validation features are used to optimize the final model using some objective function. The final model is then evaluated using the testing features and their output based on some chosen performance metric. Figure 8 illustrates this process.

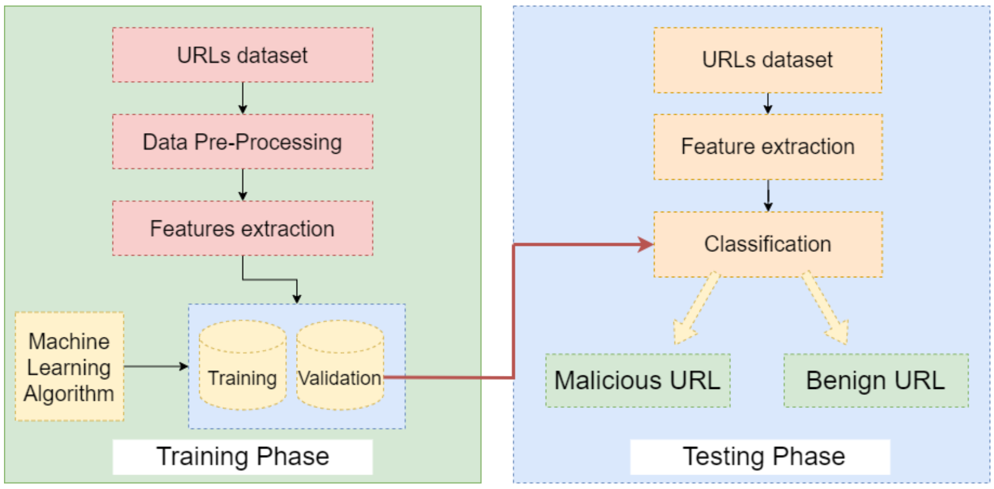


Figure 8. Supervised learning framework

2.3.2 Features and feature extraction

According to Sahoo et al. [25] a machine learning model's success critically depends on the training data's quality, which hinges on the quality of feature representation. There are four main categories of features:

- ***Blacklist features***

  A blacklist is a list that filters malicious URLs from benign ones. Despite its simplicity, it is not preferred to be used alone because of the nontrivial high false negatives and the difficulty in maintaining the exhaustive up-to-date lists.

- ***URL-based Lexical features***

  Malicious URLs have different characteristics from benign URLs. For instance, URL length is the most used feature, as malicious URLs tend to be longer than benign ones. In addition, the existence of '//' within the URL path means the user will be directed to another site. The '//' in URLs is usually located in the sixth position if the website uses HTTP and in the seventh position if they use HTTPS [26].

- ***Host-based features***

  Some features can be retrieved from the host properties of the URL. These features identify the host's location, identification, and other characteristics like IP address. URL lifetime is one of the most used features since malicious URLs have much less lifetime than benign ones. Many malicious links use the same IP host and according to Ferreira [26]. As once an IP is set, it is difficult to be changed, hence, an IP address is one of the most prominent features to indicate a malicious URL.

- ***Content-based features***

  They are "heavy-weight" features obtained from downloading the URL content. However, they lead to a better prediction model since they contain more valuable information [25].

2.3.3 Machine learning models

Machine Learning (ML) is a field of Artificial Intelligence (AI). ML models may be understood as software that has been trained to find patterns using information gathered in a variety of ways. At the same time, these models make predictions on input data and then give us output in response. Also, these models are represented as mathematical functions. At first, these models are trained over dataset, extracting and learning the pattern from those datasets. After completing the model training, they can be used to predict the unseen dataset.

In addition, there are types of ML models available based on different project goals and datasets. There are three learning models for algorithms. Each ML algorithm settles into one of the three models: the first type is supervised learning, and it is considered the simplest type among them. It has two categories: 1) Classification, 2) Regression. The second type is unsupervised learning, and the third type, which is the last type, is reinforcement learning.

The ML model we will use in this project uses supervised learning. The machine learning task is to learn a function that maps an input to an output based on example input-output pairs. It infers a function from labelled training data consisting of a set of training examples. Supervised machine learning algorithms are those algorithms that need external assistance. The input dataset is divided into train and test datasets. The training dataset has an output variable that needs to be predicted or classified. All these algorithms are famous for checkers playing programs [33].

2.3.4 Performance Metrics

With the development of performance measurement of ML models in classification problems, many methods are available that enable the evaluation of the performance of (ML) models. One evaluation method that facilitates the comparison between several models is the confusion matrices. We can define it as a technique used to determine or summarize the performance of an ML classification model for a given test dataset. Also known as error metrics. The confusion metrics have a table with four different categories. These categories are a combination of actual and predicted values.

According to [34], the four categories are True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) as follows:

- TP: number of true positives, actual malicious URLs classified correctly
- TN: number of true negatives, actual benign URLs classified correctly
- FP: number of false positives (error 1), benign URLs classified as malicious
- FN: number of false negatives (error 2) malicious URLs classified as benign

In addition, it is extremely useful for measuring the evaluation of Precision, Accuracy, Recall, and, most importantly, F1 Score. These are the equations for calculating each measurement:

Precision: This is the ratio of the positive predictions of URLs that are correctly classified [34]. It is computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Accuracy*:* This is defined as the overall success rate of the URL prediction technique[34]. It is computed as follows:

$$Accuracy\ = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

Recall: how much was correctly predicted from all the positive classes (URLs) [34]. It is computed as follows:

$$Recall\ = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

F1 Score: It is a function of precision and recall, calculated using the average of precision and recall [34]. It is computed as follows:

$$F1\, Score\ = \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

2.3.5 Related work

Over the past few years, many studies about malicious URLs using ML techniques have been conducted. Consequently, all the related work researchers used a supervised ML algorithms approach. The studies have different features, algorithms, classifiers performance, and datasets. Table 1 illustrates the comparisons between these studies, sorted from the oldest to the most recent ones.

Zeng et al. [33] discussed the malicious URLs in email content and attachment filenames to detection as a problem. This study used lexical features that can differentiate phishing URLs from benign ones selected for classification. Zeng, Y. used these models Decision Tree, Logistic Regression, Random Forest, Gradient Boosting, SVM, and GBDT. Also, the experimental results show that extracted 34 lexical features in the URL model and 15 features in the attachment model and GBDT achieved better efficiency and accuracy with > 90.71%. In addition, it got 94.36% for the URL model and attachment model. Afterward, the study explained that the lexical models are helpful in malicious email detection.

Cuzzocrea et al.[34] used decision tree algorithms to detect whether a web page exhibits phishing attacks. They considered ten host-based and lexical features, for instance, URL length, IP, and domain age. And they considered the PhishTank dataset, which consists of three classes: legitimate URLs, phishing URLs, and suspicious URLs. The researchers used five metrics to evaluate classification results: precision, recall, F-1, MCC, and AUROC. Based on their results, the best-performed algorithm was J48 in terms of precision and recall.

Divya et al.[30] proposed a method based on ML to detect malicious URLs. Also, they used the ISCXURL2016 dataset for five classes. The models used are the Lazy algorithm, Bayes net classifiers, J48 and Random Forest. Random forest outperforms all the other classifiers with > 0.961. in the experiment, they used 47 features to detect malicious URLs.

Joshi et al.[35] focused on detecting malicious URLs delivered in emails. They propose an approach using twenty-three static lexical features, for instance, URL length, numbers of '//', subdirectories, and special characters in the URL path. The lexical features were combined with one thousand trigram-based features. They collected 5 million URLs from different resources. URLs divided 60-40 between benign and malicious URLs. Based on their results, Random Forest (RF) algorithm was the best choice for the classification, with the highest accuracy and lowest False Negative Rate (FNR). In addition, they noticed that combining one thousand trigram-based along with lexical features has given the best results after comparing the performance of different feature extraction strategies that were used along with RF for classification.

Ramesh and Sab [36] study showed that Random Forest achieved the best performance by 87.85% in accuracy and misclassified 243 URLs out of 2000 predicted URLs. The dataset was obtained from the Aalto university research portal. Management process data was split individually into benign and malicious sets in the dataset. After this, they checked the sets reachability of python's who is module to ensure that features were appropriate and balancing the dataset. After they noticed that a few phishing URLs contained more than one protocol, they extracted the protocols count feature, which has never been done in any previous research work.

Johnson et al. [28] presented a public malicious URL dataset called ISCX-URL-2016. They experimented using decision tree algorithms to detect whether a website is a web-phishing site. Logistic Regression, Random Forest, SVM, Naïve Bayes, k-Nearest Neighbor, Decision Tree, Linear Discriminant Analysis, AdaBoost, Keras-TensorFlow, and Fast.ai models. To classify malicious URLs. By generating features directly from the URLs dataset. For example, domain token count, path token count, and many more were used in the experiment. They found the Random Forest performed the highest. Finally, the researcher demonstrated that the study of the specific lexical features within URLs might be used to minimize the overhead cost of a deployed model.

Alshira'H and Al-Fawa'reh [37] focused on detecting phishing URLs using lexical features. Again, they used the ISCX-URL-2016 dataset. Based on their results, Random Forest was the best algorithm with 98% Precision, Recall, and F1 Score. k - nearest neighbors' algorithm has the same results, but Random Forest outperforms in the runtime. They accounted for this outperformance for the splitting nature of the Random Forest model, which reduced the variance considerably.

Tung et al.[32] proposed discriminating various types of behavior. Also, they focused on a novel parameter set for detecting malicious URLs using ML Random Forest and Decision Tree models. They used binary classification to classify the URLs into benign and malicious URLs. Random Forest indicated an accuracy of over 97% outperforming the rest.

Wejinya and Bhatia [38] proposed MuD (Malicious URL Detection) model applied to three machine learning classifiers, support vector machine (SVM), logistic regression, and Naïve Bayes. The last classifier recorded the highest Recall, Accuracy, F1 Score, and Precision by 100% for each of the 5-evaluation metrics. The model was trained using an existing dataset from the UCI machine learning repository, which contains 11.055 URLs with fifteen unique features. Features are from unique feature categories, for instance, they used URL length as a lexical feature. In the host-based features part, they check if the URL has an associated IP address. Also, they use some content-based features.

Ahammad et al. [39] investigate the URL's fifteen lexical and domain-based features. They consider the training and testing accuracy as the performance evaluators. LightGBM has the best accuracy of 86%. Though there are fifteen features, only some are essential in increasing accuracy. Based on the comparison, URL length was the most important feature.

Bhanuprakash et al. [30] have focused on detecting malicious posts and campaigns of apps on Facebook. They used FRAppE, arguably the first tool. The dataset used was gathered by observing the posting behavior of users on Facebook. The researchers have identified a set of features and then leveraged these distinguishing features. The feature extraction component has three subcomponents: grouping identical domains, finding entry point URLs and extracting feature vectors. Finally, the FRAppE tool got a percentage of 99.5% accuracy in detecting malicious apps.

Alsaedi et al. [31] In this study, they used cyber threat intelligence (CTI) extracted from Google and Whois websites to develop a model that detects malicious websites. Also, they have been used (CTI) as an effective and safer alternative to improve the detection accuracy of malicious websites using two-stage ensemble learning. Also, they used the Random Forest model for pre-classification with multilayer perceptron for final decision-making. Results show that the CTI-based model achieved a 7.8% accuracy.

Table 1 summarizes all the reviewed studies and provides more details on the chosen classifier's performance for each study. Because some studies do not calculate recall, Table 1 has been divided into two parts, the studies that did calculate recall—the measure we focus on—and the studies that depended on accuracy. As some of the classifiers are more popular, Figure 9 shows the classifiers distribution among the reviewed literature.

Table 1. Related work comparison

Paper	**Year**	**Features Extraction**	**Classifiers Performance**		**Dataset**
			Recall		
Cuzzocrea et al. [34]	2019	10 host-based and lexical Features	J48 HoeffdingTree (HT) Decision Stump (DS) Random Forest (RF) LMT Rep Tree (RT)	91.6% 91.6% 91.3% 91.2% 90.5% 87.2%	1353 URLs, 548 legitimate URLs, 702 phishing, and 103 suspicious
Divya et al. [30]	2019	Features are already in the dataset.	Random Forest Lazy Classifier (LC) J48 Bayer-Net (BN)	96.1% 95.4% 94.4% 92.2%	114250 samples. 5 URL classes such as benign, defacement, malware, phishing, and spam.
Johnson et al. [28]	2020	78 static lexical analysis	Random Forest (RF) Decision Tree (DT) k-Nearest Neighbor (KNN) Keras-TensorFlow (KTF) Fast.ai AdaBoost (AB) Linear Discriminant (LD) SVM Logistic Regression (LR) Naïve Bayes (NB)	98.68% 97.63% 97.47% 97.13% 96.88% 96.21% 94.43% 93.96% 90.50% 68.73%	18982 samples. 5 URL classes such as benign, defacement, malware, phishing, and spam.
Alshira'H and Al-Fawa'reh [37]	2020	Around 80 lexical features.	Random Forest (RF) KNN Decision Tree (DT) SVM Logistic Regression (LR) Perceptron Quadratic Discriminant (QD) Gaussian Naive Bayes (GNB)	98% 98% 97% 96% 95% 89% 77% 77%	45k URLs, 35k benign URLs, and 10k for phishing URLs.
Wejinya and Bhatia [38]	2021	15 lexical, host-based, content- based features.	Naïve Bayes SVM Logistic regression	100% 95% 90%	UCI machine learning repository having 11,055 URLs both malicious and benign.
Alsaedi et al. [31]	2022	Lexical feature.	Random Forest Decision Tree Sequential deep learning Convolutional neural network Logistic Regression Naive Bayes	96.88% 95.71% 95.80% 94.91% 90.82% 88.73%	651191 URLs Various.

Paper	Year	Features Extraction	Classifiers Performance		Dataset
			Accuracy		
Zeng et al.[33]	2018	lexical features	GBDT Random Forest (RF) SVM Decision Tree (DT) Logistic Regression (LR)	90.71% 90.4% 89.80% 88.8% 84.6%	15000 benign URLs and 15000 phishing URLs, that is 30000 samples in total
Joshi et al. [35]	2019	23 static lexical features	Random Forest (RF) Gradient Boost (GB) AdaBoost (AB) Logistic Regression (LR) Naïve Bayes (NB)	92.0% 90.0% 90.0% 87.0% 70.0%	5 million URLs spitted 60% benign and 40% malicious.
Ramesh and Sab [36]	2019	Dynamic Features and Lexical Features.	Random Forest (RF) Decision Tree (DT) KNN Kernel SVM SVM Logistic Regression (LR) Naïve Bayes (NB)	87.9% 86.8% 86.6% 84.5% 77.7% 75.4% 69.0%	95913 URLs. Sampled into 5000 URLs for each benign and phishing
Tung et al. [32]	2021	Set of 14 lexical and host-based	Random Forest (RF) Decision Tree	97.49% 96.33%	68951 URLs Various.
Ahammad et al. [39]	2021	15 lexical domain-based features	LightGBM Random Forest Decision Tree Logistic Regression SVM	86% 85% 85% 84% 83%	3000 URLs, splitted 50% for each malicious and benign
Bhanuprakash et al. [30]	2022	The feature extraction component has three subcomponents	Support Vector Machines (In FRAppE tool)	-	-

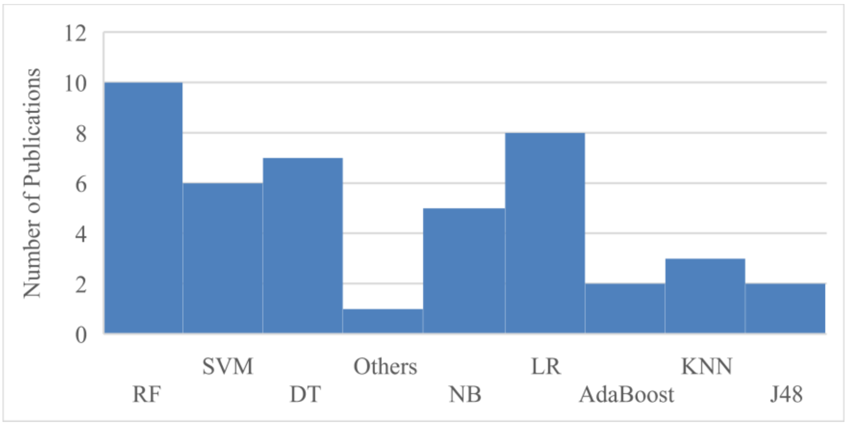


Figure 9. The number of publications in the reviewed literature for each classifier.

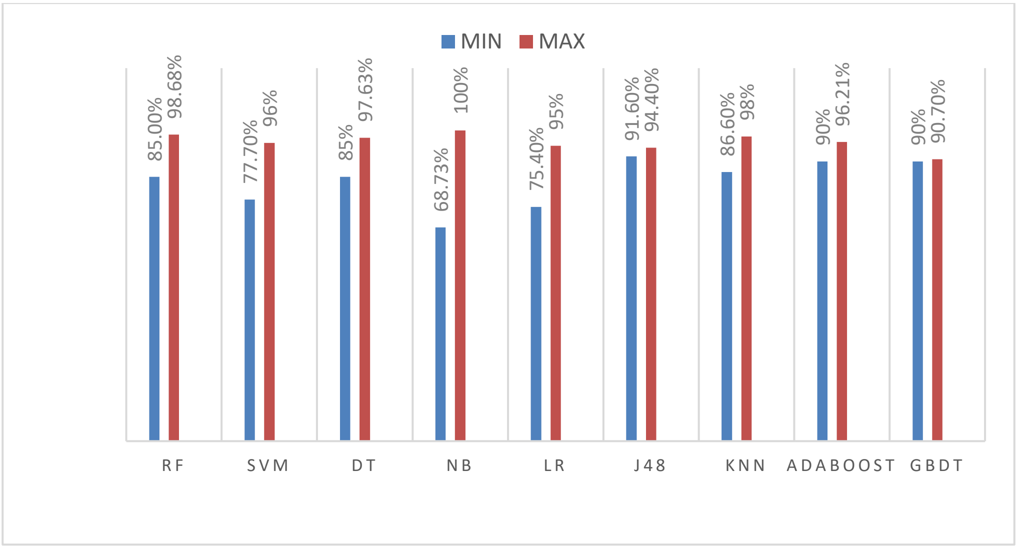


Figure 10. Performance of Classifiers

Figure 10 above shows how many scientific papers have used these classifiers to create a model. It turns out that (RF) model is the most used in most research. Followed by algorithms (DT) and then (SVM). Figure 10 shows the performance of classifiers by the ratios of all the classifiers. It was found that the (NB) classifier outperformed the maximum accuracy rate, which was 100%, while it comes after it with a percentage close to it, which is the (RF) classifier, its maximum accuracy rate was 98.68%, as for the rest of classifiers, its range of maximum from 90% - 98%. As for the best minimum accuracy ratio, it got J48 with 91.60%.

In general, it is clear from the results that (RF),(DT), (NB),(KNN) and SVM are the most appropriate model for deployment in a malicious URL detection system.

2.4 Summary

The basic knowledge for customizing an Android system with an embedded ML-based malicious URL detector was covered. This review includes the architecture of the Android system and its main layers and components. In addition, we went through the system's security, features, and some limitations of the system. We found that the Android system has lower security than its competitors.

After that, we covered the URLs in general, their components in detail, and how they can be used for theft, plagiarism, and other electronic crimes by hackers. We also discussed ways to identify and avoid malicious links and the tools that help protect against them. However, none of these tools provide system-wide protection against malicious URLs.

One method to detect malicious URLs was using a ML model. Hence, this chapter briefly discussed the primary supervised learning framework; its stages have been detailed and explained. And we identified ML models and ML model types. Also, we defined the most popular method to evaluate ML models: confusion metrics. Finally, we discussed the related work and compared them in Table 1.

Chapter 3 – SYSTEM ANALYSIS

This chapter talks about the new system's requirements, including (functional and non-functional requirements) and then exposes our solution and other alternatives.

3.1 System requirements

3.1.1 Clients, customers, and users

Applications shall access the system on behalf of the end users.

3.1.2 Functional and data requirements

User Requirements: Applications must access the system to detect malicious URLs

System Requirements:

1.1 A machine learning model that detects malicious URLs.

1.2 High-accuracy model

User Requirements: Enable the user to override if the user clicks on malicious URLs.

System Requirements:

2.1 The system prevents user access to malicious URLs.

2.2 The System alerts the user in case the URL is malicious.

2.3 The System gives information about the URL.

2.4 The system gives the user freedom to choose whether to continue or cancel.

User Requirements: Authorization levels: determines the level of data access

System Requirements:

3.1 the system asks the user to access the operating system and process the chosen option.

3.1.3 Non-functional requirements

- Secure

1.1 The system will protect the user from malicious web URLs.

- Available

2.1 System must be available 24/7.

- Reliable

3.1 Notify the user with alert messages.

- Robust

4.1 System will adapt to new malicious URLs with new features

3.1.3.1 Security requirements

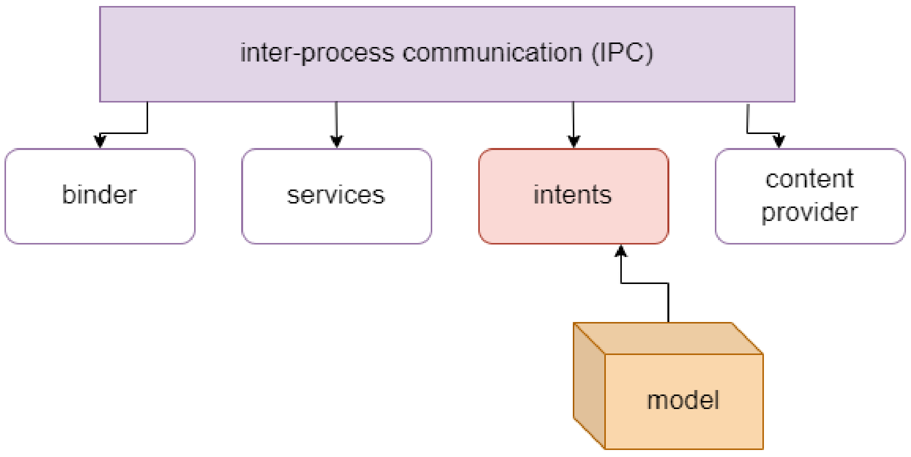
The system shall not allow any malicious URL to cross. And shall be safe to use.

3.1.3.2 Performance requirement

The models shall have a high recall and be able to detect more than 90% of the malicious URL.

3.2 Proposed Solution

We will customize the android system and add a machine learning model component, so any detection will be based on this newly added component.



.

Figure 11. we will link our model with the intent class

3.3 Alternative Solutions

3.3.1 Most similar application to our proposed solution

There is no Android system completely identical to ours, but there is an application that does a lot of things besides checking malicious URLs, which is the most similar to ours:

WOT Mobile Security Protection [40]

An Android application keeps the user safe against potential threats and scams by using accessibility permissions to see the requested URL. It blocks the requested URL in real time if you enter a malicious or unsafe website. Unlike our solution, their application is not integrated with the Android device OS. So, it requires a human-maintained list of malicious websites to verify that any web link is free of any malicious code.

3.3.2 Other solutions

There are other alternative solutions that are available, such as antivirus applications for the Android system that protect against several threats, and some of them provide a service to verify the integrity of the URL from any threats, such as:

Sophos Intercept X for Mobile[24]

According to [41]. It is an antivirus app for Android that scans apps for malware and harmful content while they are being installed and notifies you of any possibility of leaking sensitive data. Part of Sophos' features is the Secure QR Code Scanner, which checks any target URL for potential threats or malicious activity when scanning QR codes.

The URL checking service here is limited to the QR code only. Unlike our system, which is integrated into the OS and specializes in web links, and automatically checks all web links on the various platforms used by the user, which provides greater protection from the threat of malicious.

Also, there are applications we can get from Google Play dedicated to checking only the URL, such as:

Link Protector: URL Security [42]

It is an Android application a security service with insecure link protection, website detection, and website blocker It scans, detects, and blocks all types of fake, spam, and insecure links and websites.

Link Protector in accordance with [42]. comes with features like URL Shortener, Favorite Link Manager, and Realtime Website Blocker. Unlike our system, which is integrated with the Android device OS, which in turn will automatically verify that any web link is free of any malicious code without human intervention on any platform, not only on the browser.

Scan URL - Link Checker[43]

An Android app that allows website security checks to protect against phishing, malware, and blacklisted websites. It requires the user to manually verify the links by copying and pasting them into the application to verify them, unlike our system, which automatically checks the links as it provides this service without any effort from the user. As stated in [43]

And other applications we can download from external sources like get-hub:

Android-Application-Malicious-URL-Detector[44]

It is an external-source Android application that is not supported by Google Play. It requires the user to copy and paste the link manually to verify that it is free of any malicious code, unlike our system, which is integrated with the Android device OS, which in turn will automatically verify that any web link is free of any malicious code without human intervention.

3.3.3 Features Summary

Table 2. Alternative Solutions Features

	Detect malicious apps	Detect malicious URLs	Website blocker	Secure QR Code Scanner	ML	Build in the system
Sophos Intercept X for Mobile[24] [41]	√	√		√		
Link Protector: URL Security[42]		√	√	√		
Scan URL - Link Checker[43]		√				
Android-Application-Malicious-URL- Detector[44]		√				
WOT Mobile Security Protection [40]	√	√	√		√	
Our Android		√			√	√

3.4 Summary

This chapter introduced the system requirements, the customized system's solution, and other alternative solutions that can be applied.

Chapter 4 – DESIGN CONSIDERATIONS

This chapter talks about the design constraints which are the hardware and software environments that the system needs, then give more details about the algorithm that will be used, how will we reuse the existing software parts, the methodology that has been used to develop this project at the end we expose our future.

4.1 Design Constraints

4.1.1 Hardware and software environment

Hardware environment:

The system will be compatible with all android hardware. The minimum hardware requirements for the system are as follows:

- Capacity 32GB or more.
- Operating system Android 11.
- Processor speed 1.5 GHz
- 2 GB RAM or more.

Software environment:

- The operating system will be based on the Linux kernel.
- The system will be available for most of the well-known web browsers and applications that are compatible with android OS.

4.2 Architectural Strategies

4.2.1 Algorithm to be used

In this project, after reading several research studies about the classification models used to detect malicious URLs. We found that the three modes widely used in this field with the most effective results are RF, NBC, and SVM. These Machine Learning algorithms will be employed for detecting malicious URL dataset classification.

In addition, these algorithms are widely used in practice for supervised learning purposes. Also, these algorithms have been shown to perform well for such classification problems.

4.2.2 Reuse of existing software components

The entire system will be reused. The component that we will focus on is one of the IPC mechanisms, which is called intent.

4.2.3 Development method

We will develop this project using the waterfall methodology, a sequential development process that flows like a waterfall through all phases of a project. The main phases are analysis, design, development, and testing. Each phase is completely wrapped up before the next phase begins. [45]

4.2.4 Future enhancements/plans

We can develop the system to detect other malware and build intelligent malware detection.

4.3 Summary

This chapter introduced the design constraints (hardware and software environments), gives more details about the used algorithm, how will we reuse the existing software parts, the development methodology, and the future.

Chapter 5 – SYSTEM DESIGN

In this chapter, we will discuss our android design, and how its structure, interacts and behaves when it detects malicious URLs.

5.1 Behavioral diagram: Activity diagram

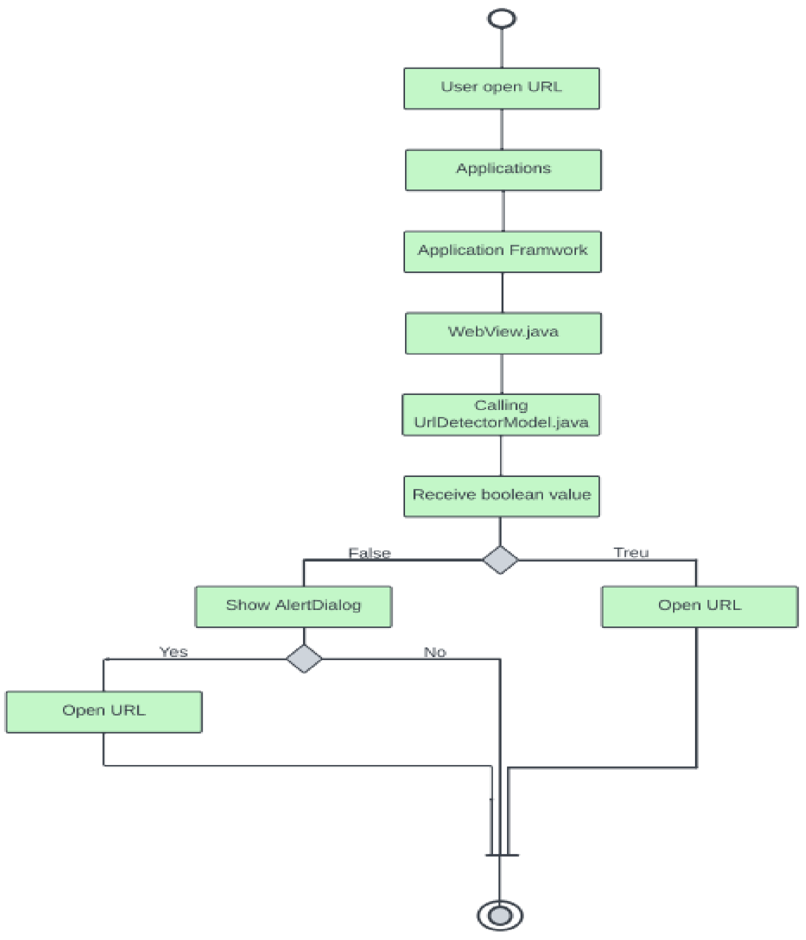


Figure 12. Activity Diagram

In Figure 12, When the applications layer reserves a URL, it will contact the application framework layer to get all the resources it needs to open this URL.

In the application framework layer, there is a WebView class it contains the LoudUrl function. which checks the URL that the applications layer wants and sends it to our UrlDetectoModel to check if it is a malicious or benign URL. If our model returns true means it benign URL and if the results are false means it's a malicious URL. And it will show an alert dialog to the user and let him make his own decision whether to continue opening the URL or cancel.

5.2 Structural diagram: Component diagram

In Figure 13, We have the applications layer that receives the URL requests from the user. Then it will send it to the application framework layer that has the WebView class. which is responsible for displaying Any URL and our UrlDetectorModel class which decides whether the URL is malicious. So an alert dialog will be sent or not. Finally, The application framework will communicate with the Linux kernel component to request any needed resources.

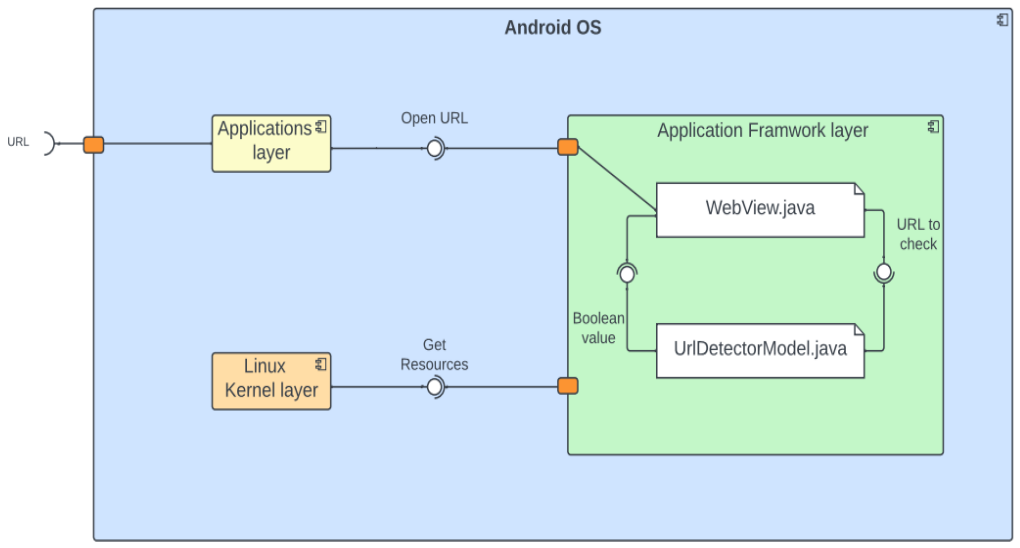


Figure 13. Component Diagram

5.3 Interaction diagram: Sequence diagram

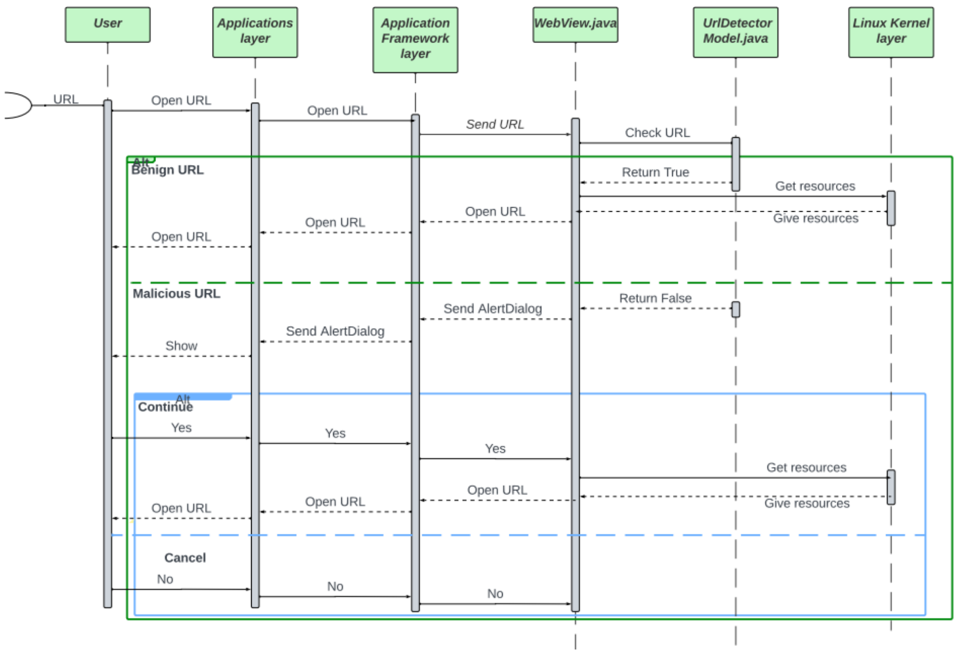


Figure 14. Sequence Diagram

In Figure 14, the application layer communicates with the application framework layer to open the URL by using WebView class and then sent a check request to UrlDetectorModel class.

If the Model finds the URL benign will returns true to the WebView class the Linux kernel layer will accept it and goes back to get all the needed recourses, then the URL will be opened.

Otherwise, if the Model finds out that the URL is malicious an alert Dialog will appear and the user can make his own choice to open the URL or just cancel.

Chapter 6 – IMPLEMENTATION

This chapter explains in detail our ML modeling methodology including the extracted features, performance metrics, and configuration. It also includes the dataset and tools used in our experiment. Likewise, we will cover all the steps of implementation, and the results of the algorithms used in the ML model, and compare them.

6.1 Developing ML models

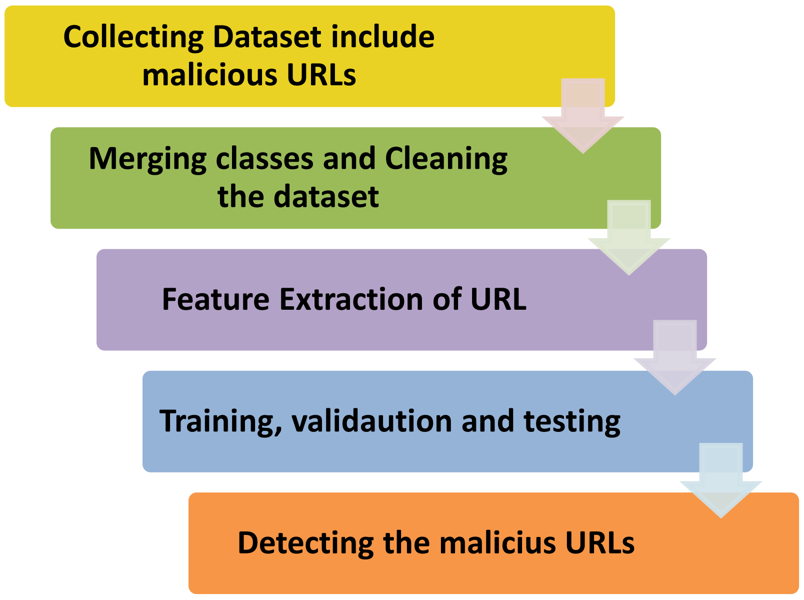

Collecting Dataset include malicious URLs
Merging classes and Cleaning the dataset
Feature Extraction of URL
Training, validaution and testing
Detecting the malicius URLs

Figure 15. Methodology of the Model Processes

As shown in Figure 15, the methodology of the model processes. The first step in the experiment took a number of samples as a dataset. Then we cleaned the dataset and merged the four classes so that we only have two classes of malicious and benign URLs. After that, we extracted the features from the dataset. Then, we divided the dataset into 10% for testing, 20% for validation, and 70% for training. The experiment was by using three classifiers and the dataset, which has 78909 samples of various URLs. There were 35873 benign URLs and 43036 malicious URLs. Also, build the model (RF), (SVM), and (NB) algorithms. Therefore, the model was trained to classify malicious URLs from the dataset. To provide satisfactory results. Emphasis has been placed on the importance of high accuracy and recall. The model has been tested with the previously mentioned classification algorithms. Very good results have been obtained through this experience.

6.1.1 ML models

This subsection covers the most popular supervised ML models used for classification. Many types of models can do tasks. Every task usually requires a different model because each one is used to perform a specific classification problem.

The list of most popular or traditional (ML) classification models tested in this project are:

Random Forest (RF)

RF is a supervised ML algorithm, and a classification algorithm consisting of a collection of Decision Trees. this algorithm or model is very convenient for high-dimensional data modeling; it can handle binary data. As for the way it works, it uses features like randomness and bagging to build each individual tree to create an uncorrelated forest of trees. so that it can be predicted.

Naive Bayes Classifier (NBC)

It is one of the supervised powerful (ML) algorithms that is used for classification. It is based on the Bayes theorem with the “naive” that is finding the probability of an event after it occurred.

Decision Tree (DT)

Decision Tree Classifier is a class capable of performing multi-class classification on a dataset [37].

Logistic Regression (LR)

This classification algorithm usually gives a properly calibrated probability and can be used in categorizing whether a URL is malicious or not since it only supports binary classification problems [48].

Support Vector Machines (SVM)

SVM is a supervised ML algorithm that may be used for both classifications based on its strong foundation in positive results obtained in several research that we read it so it is capable of performing classification very effectively by a statistical learning-based approach, which is drawing a straight line between two classes by linear SVM.

6.1.2 Resources

As mentioned before, the project aims to detect malicious URLs. To provide a robust and high-efficiency knowledge base for training the model. We have obtained 114,400 URLs as our initial dataset from (ISCX-URL2016) [49].

The model will be trained and tested on the Python 3.10.5 system using Jupyter Notebooks. The most important Python libraries that will be used to train the (ML) model are NumPy, Scipy, Scikit-learn, Theano, TensorFlow, Keras, PyTorch, and Pandas.

This section explore this dataset and its features. Also, some of the features were redefined as they were reversed engineered.

6.1.2.1 Dataset analysis

This dataset is unbalanced, and we will work on making it balanced, and it contains five types of URLs containing benign and malicious URLs. The four categories of malicious are Phishing, Spam, Defacement, and Malware. We will combine the four classes into one class to represent malicious URLs. To finally be ours, two classes, Malicious and benign URLs. Also, we will clean and merge datasets according to the model build needs. Table 1illustratesthe number of samples in the dataset.

Table 3. Types of datasets and number of samples

Type of URL	**Number of samples**
Benign	35,300
Spam	12,000
Phishing	10,000
Defacement	45,450
Malware	11,500

Studying the distribution and analyze the variables. we import the seaborn library and use the method countplot then we got the results in Figures (16, 17, 18, 19).

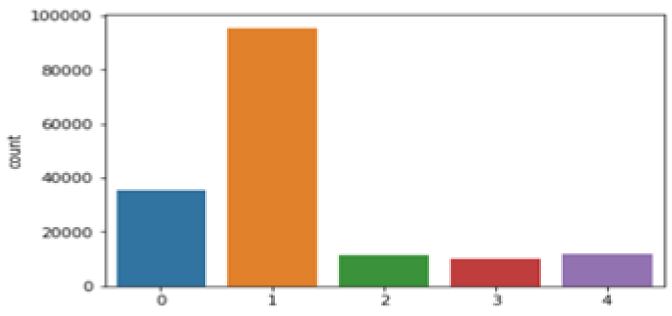


Figure 16. Plot for variable Y

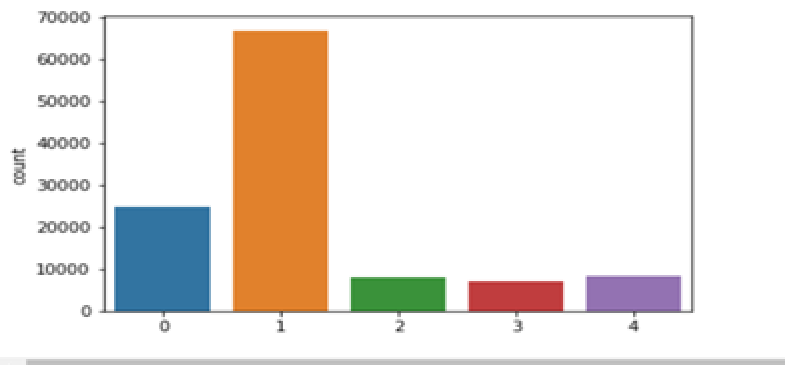


Figure 17. Plot for variable y_train

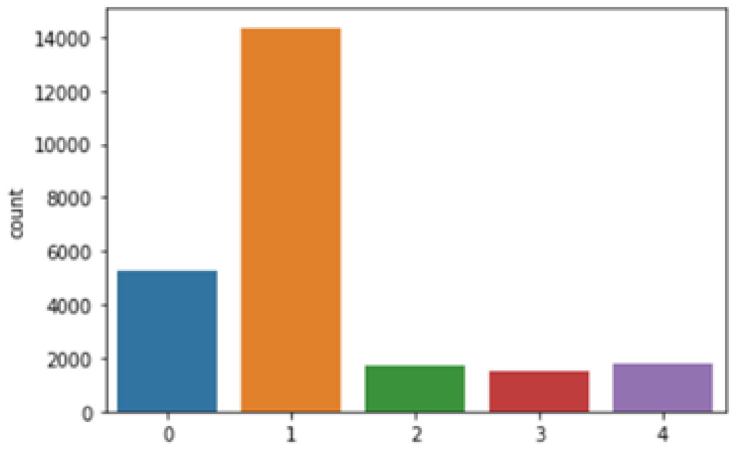


Figure 18. Plot for variable y_test

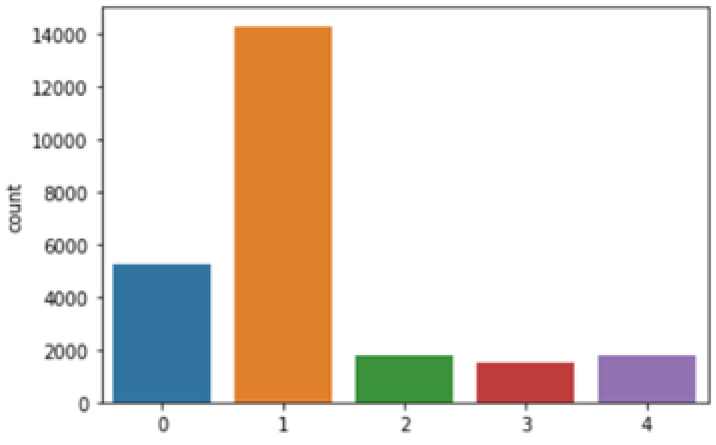


Figure 19. Plot for variable y_valid

6.1.2.2 Data processing

So, we use mutual information as features selector and it helps us to arrange the data in descending order from most important to the least one, we found out that there are none important 13 features so we dropped them

After dropping the features, the number of null values decreased into 349 therefore it was possible to remove the null values without the dataset decreasing dramatically.

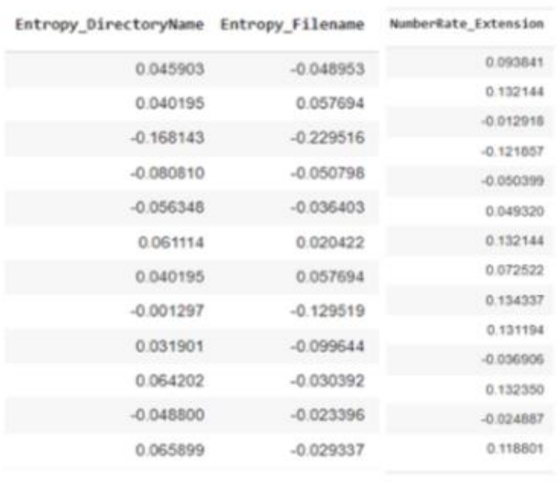
Our dataset includes a categorical column called ‘URL_Type_obf_Type’, so it was necessary to replace it with numerical values using hot encoding. This column has 4 unique values: Defacement ,benign ,malware, phishing, spam. We replaced 'benign' with 0, 'Defacement' with 1, 'malware' with 2, 'phishing' with 3, and 'spam' with 4. The following python code snippet was used:

```
import pandas as pd

dataset = pd.read_csv("/content/All.csv")

#your old value(string) and value is your new value(integer).

URL_Type_obf_Type = {'Defacement': 0, 'benign': 1, 'malware':
2,'phishing':3,'spam':4}

#Assign these different key-value pair from above dictiionary to your table

dataset.URL_Type_obf_Type = [URL_Type_obf_Type[item] for item in
dataset.URL_Type_obf_Type]

#New table

dataset.sort_values(['URL_Type_obf_Type'],axis=0,
ascending=True,inplace=True,na_position='first')

print(dataset)
```

After getting a dataset with columns, all of numerical characters now it’s time for cleaning. The number of null values in the dataset was 19183 before the cleaning.

The features that had the most null values are NumberRate_Extension , Entropy_DirectoryName, Entropy_Filename, which had the values 10130, 8468, 236 respectively.

A correlation study was made to check if dropping these features would affect the results, and since the study shows that the features had no correlation - negligible correlation that wouldn’t affect the results therefore the features were dropped.



Entropy_DirectoryName	Entropy_Filename
0.045903	-0.048953
0.040195	0.057694
-0.168143	-0.229516
-0.080810	-0.050798
-0.056348	-0.036403
0.061114	0.020422
0.040195	0.057694
-0.001297	-0.129519
0.031901	-0.099644
0.064202	-0.030392
-0.048800	-0.023396
0.065899	-0.029337

NumberRate_Extension
0.093841
0.132144
-0.012918
-0.121857
-0.050399
0.049320
0.132144
0.072522
0.134337
0.131194
-0.036906
0.132350
-0.024887
0.118801

Figure 20. Using the correlation method in python to check if removing the features with the most null values would affect the results.

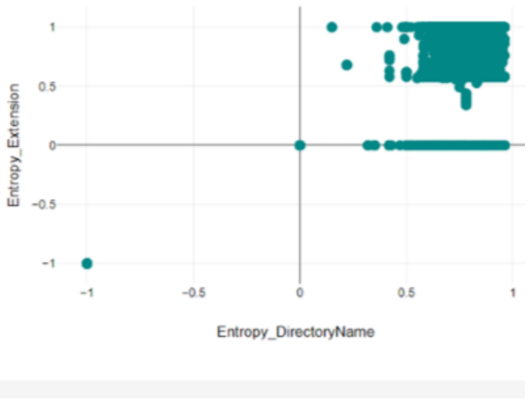


Figure 21. Shows an example of no correlation between Entropy_DirectoryName and Entropy_Extention.

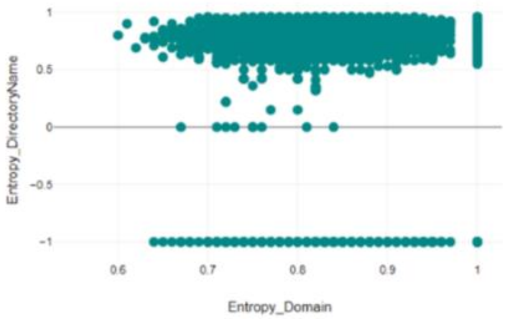


Figure 22. Shows an example of no correlation between Entropy_DirectoryName and Entropy_Domain

As shown previously, the dataset suffers from class imbalance. To make it balance we needed to choose 10000 malicious URLs from the classAs defacement (class 1) by using method Sample ,then we merged the 4 malicious classes (1,2,3,4) to one binary class giving 0 to benign and 1 to malicious and we got this result in Figure 23.

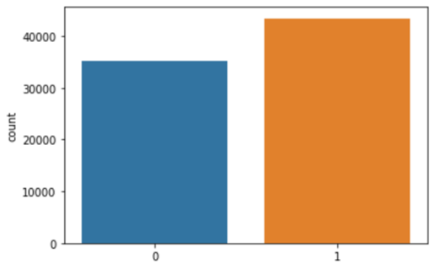


Figure 23. Plot for variable y after choosing 10000 malicious URLs from the class defacement (class 1) by using method Sample ,then we merged the 4 malicious classes (1,2,3,4) to one binary class giving 0 to benign and 1 to malicious .

6.1.3 Feature extraction

Since lexical features have lightweight computation, safety, and high classification accuracy, they have become one of the most popular sources of features in machine learning [48].

Feature set 1: Large

The used dataset presents around 79 lexical extracted features from URLs. For instance: URL length, domain length, and the number of dots in the URL. Table 3 samples list the features of the dataset.

Table 4. Feature Set 1: Large

'Querylength'	'domain_token_count'	'path_token_count'
'avgdomaintokenlen'	'longdomaintokenlen'	'tld'
'avgpathtokenlen'	'charcompvowels'	'charcompace'
'ldl_url'	'ldl_domain'	'ldl_path'
'ldl_filename'	'ldl_getArg'	'dld_url'
'dld_domain'	'dld_path'	'dld_filename'
'dld_getArg'	'urlLen'	'domainlength'
'pathLength'	'subDirLen'	'fileNameLen'
'this.fileExtLen'	'ArgLen'	'pathurlRatio'
'ArgUrlRatio'	'argDomanRatio'	'domainUrlRatio'
'pathDomainRatio'	'argPathRatio'	'executable'
'isPortEighty'	'NumberofDotsinURL'	'ISIpAddressInDomainName'
'CharacterContinuityRate'	'LongestVariableValue'	'URL_DigitCount'
'host_DigitCount'	'Directory_DigitCount'	'File_name_DigitCount'
'Extension_DigitCount'	'Query_DigitCount'	'URL_Letter_Count'
'host_letter_count'	'Directory_LetterCount'	'Filename_LetterCount'
'Extension_LetterCount'	'Query_LetterCount'	'LongestPathTokenLength'
'Domain_LongestWordLength	'Path_LongestWordLength'	'subDirectory_LongestWordLength'
'Arguments_LongestWordLength'	'URL_sensitiveWord'	'URLQueries_variable'
'spcharUrl'	'delimeter_Domain'	'delimeter_path'
'delimeter_Count'	'NumberRate_URL'	'NumberRate_Domain'
'NumberRate_DirectoryName'	'NumberRate_FileName'	'NumberRate_Extension'
'NumberRate_AfterPath'	'SymbolCount_URL'	'SymbolCount_Domain'
'SymbolCount_Directoryname'	'SymbolCount_FileName'	'SymbolCount_Extension'
'Entropy_URL'	'SymbolCount_Afterpath'	'Entropy_Domain'
'Entropy_DirectoryName'	'Entropy_Filename'	'Entropy_Extension'
'Entropy_Afterpath'		

The following figure shows the range of the feature importance for these 79 features based on mutual information criterion.

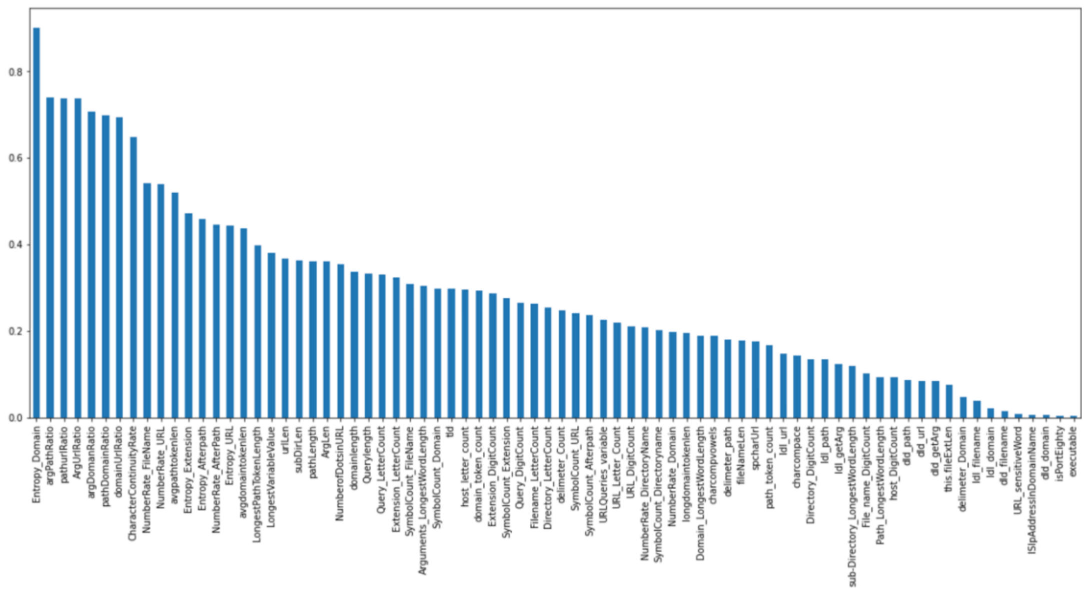


Figure 24. Finding the important features by using mutual information as features selector

Feature set 2: Small

We have written a code to extract 11 features of URLs features completely by java. Code takes the dataset file consisting of 2 columns URLs and labels, and the output is 3 files output.csv have the result features and labels, outputUrl.csv have the URLs, result features, and labels, and the last file is the exception file, it has the URLs that cause exceptions. We used java.net.URl and its functions [52]. Table 5 shows features extracted from URLs.

Table 5 : Feature Set 2: Small

'urlLen'	'pathLength'	'Querylength'
'URL_Letter_Count'	'URL_DigitCount'	'host_letter_count'
'host_DigitCount'	'SymbolCount_URL'	'spcharUrl'
'Query_DigitCount'	'pathurlRatio'	

As we did with the large feature set, we performed an analysis for the feature importance using the information mutual criterion. The ranking is shown in the following Figure 25.

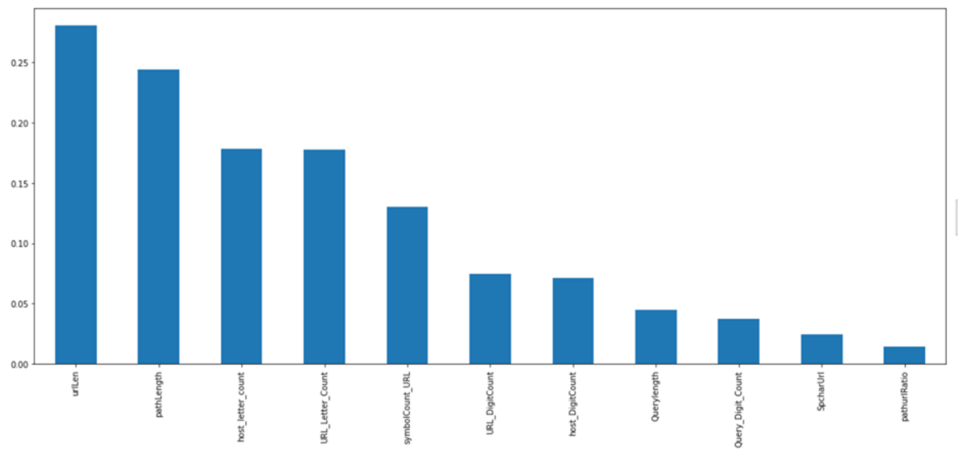


Figure 25. The importance of the features.

6.1.4 Experiment design

For the experiment, the dataset of URL samples will be divided into two classes: benign and malicious. In addition, we divided the dataset into three groups: 10% for testing, 20% for validation, and 70% for training. Three classification algorithms (RF, SVM, and NBC) will be used in the experiment, and we will build models using these algorithms.

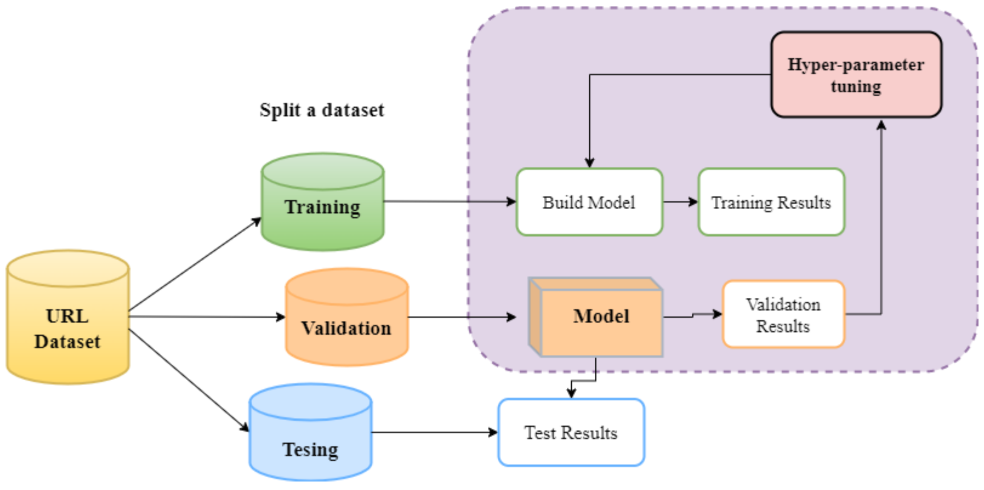


Figure 26. The Model building phases

In the experiment, as shown in Figure 26, we have three phases to build the model. Initially, it will split the dataset into three subsets. one for training, the second for validation, and the last set for testing the model. Then it will be trained using the URL dataset to build the model. After building the model. it will start the validation phase to validate the model during the training process and its results. the results of the validation process assist us in changing parameters to provide satisfactory results and optimize the model. In the last phase, it will be tested using the URL dataset the model didn't train on to provide high accuracy, precision, and performance of the model created during the training phase.

First, we assigned all the features to the variable X and the column 'URL_Type_obf_Type' which includes the labels to variable Y. Then we split it the to testing and training sets.

Now we will train the model using the three algorithms (RandomForest, Naive Bayes, SVM) to find out the most efficient one. We used sklearn python library. The following parameters were used for each model.

RandomForest	**Gaussian NB**	**SVM**
n_estimators=100	*priors=None*	*penalty='l2'*
criterion='gini'	*var_smoothing=1e-09*	*loss='squared_hinge'*
max_depth=None		*dual=True*
min_samples_split=2		*tol=0.0001*
min_samples_leaf=1		*C=1.0*
min_weight_fraction_leaf=0.0		*multi_class='ovr'*
max_features='sqrt'		*fit_intercept=True*
max_leaf_nodes=None		*intercept_scaling=1*
min_impurity_decrease=0.0		*class_weight=None*
bootstrap=True		*verbose=0*
oob_score=False		*random_state=None*
n_jobs=None		*max_iter=1000*
random_state=None		
verbose=0		
warm_start=False		
class_weight=None		
ccp_alpha=0.0		
max_samples=None		

6.1.5 Results

Feature set 1: Large

The experiment demonstrates the classification of malicious URLs. Where URLs are a binary classification with two classes: benign (0) and Malicious (1), the experiments utilize the machine learning models the results shown in Figure 27.

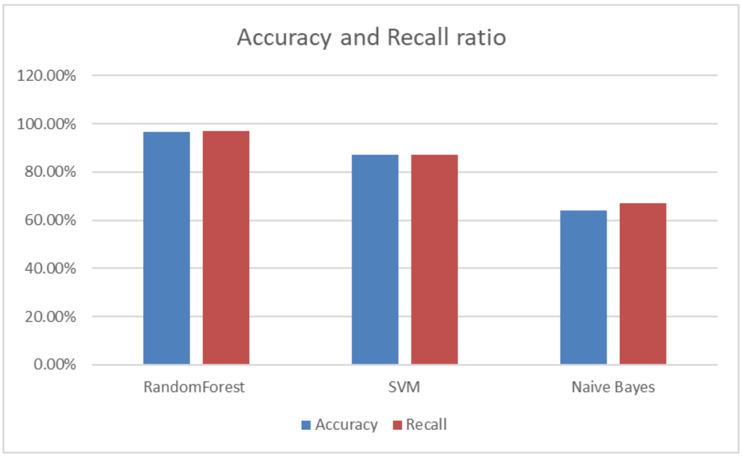


Figure 27. Algorithms Accuracy and recall Ratio

Figure 27 shows the accuracy and recall ratios of all three algorithms used to train the model. Based on the gotten result the RandomForest algorithm has the highest accuracy and recall rate with 96.7% of accuracy and 97% of recall, then we have a close result with the SVM algorithm with 87% of accuracy and 87% of recall, and the lowest accuracy and recall went to Naïve Bayes with 64% of accuracy and 67% of recall.

As the outperforming model was RF, we list its confusion matrix in Table 7.

Table 7. Confusion matrix using RF

Predicated	*0*	*1*	*2*	*3*	*4*
Actual 0	1179	6	0	33	1
Actual 1	0	1152	3	12	0
Actual 2	2	2	996	17	1
Actual 3	12	15	2	1109	6

Feature set 2: Small

Based on the previous experiments, Random Forest (RF) model was satisfying to us to build our model using it. Hence, we use to develop the models using the small feature set. Table 6 shows the number of positive and negative states of TP, FP, FN, and TN. Which are details about the accuracy, error, and precision ratio of the RF algorithm. We found the RF classifier was the best choice for the classification. Given that it has the highest precision and lowest FPR among other classifiers.

Table 6. Confusion matrices by using RF trained on Feature set - Small.

	Predicated 0	**Predicated 1**
Actual 1	5065	157
Actual 0	249	6324

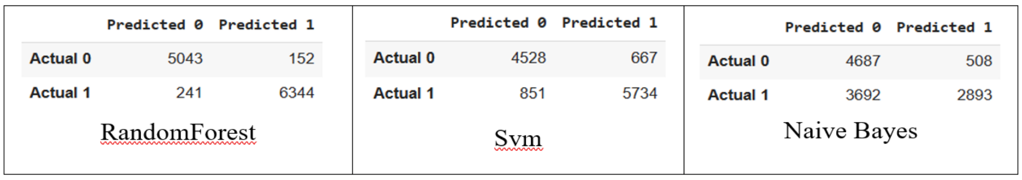
After getting the result we had some problems with deployment the model so needed to write the model and extracting the features in java language.

6.1.6 Discussion

The present study sought to help social media users detect malicious URLs. Toward this end, it compared three classification algorithms RandomForest, Naive Bayes, and SVM. The results demonstrated that the RandomForest algorithm had higher accuracy than the Naive Bayes and SVM algorithms (>96%).

By collecting the accuracy, recall, and confusion metric results of the algorithms in conjunction with the dataset, we conclude that despite the popularity of the Naive Bayes and SVM algorithms, they are inferior to RandomForest due to the higher variance of their results. In general, it is clear from the study results that RandomForest is the most appropriate model for deployment in a malicious URL detection system.

The RandomForest algorithm demonstrated higher accuracy than all the other algorithms RandomForest(>96%), Svm (>87%), Naive Bayes(>64%), and recall with RandomForest(>97%), Svm (>87%), Naive Bayes(>67%) (. In addition, with regard to its confusion metrics, as shown in Figure 28.



	Predicted 0	Predicted 1
Actual 0	5043	152
Actual 1	241	6344

RandomForest

	Predicted 0	Predicted 1
Actual 0	4528	667
Actual 1	851	5734

Svm

	Predicted 0	Predicted 1
Actual 0	4687	508
Actual 1	3692	2893

Naive Bayes

Figure 28. Confusion matrices of the three algorithms RandomForest, SVM, and Naïve Bayes.

However, SVM is also a very good classifier as its results were close to those of the RandomForest algorithm. but, In the present study, we will use RandomForest.

6.2 Deploying ML model to Android OS

After completing the ML model building via python language. We found out that we should export the model from python language to java language. So that we can put the model inside the Android system.

6.2.1 Using m2cgen library

To export the model from python to java language, we have taken several steps. Described as follows:

1. We have used the **m2cgen** library to export the model from python to java language [51].
2. The second step was to save and load the ML model.
3. After the previous step, we used this code to export our model to java: `model_to_java = m2c.export_to_java(classifier)`
4. The last step was to save the ML model in java language.

The model have four functions: score(), subroutine0(), addVectors(), and mulVectorNumber() . The problem here is that score() is consist of more than 931k lines and subroutine0() have 19k lines. A Java had a size limit for its methods which is 64k bytes. Our generated model exceed this limit.

We used some solutions like run code in different IDEs and in terminal, but the problem not solved. Hence, the only solution given this model is to refactor the model file and break down the score and subroutine0 methods into smaller nested methods, which takes a very long time and effort to be accomplished.

6.2.2 Using JPMML

As the converted model via m2cgen library would not be compiled due to the limitation of the method size, another method was explored. Here, the model was exported into a Predictive Model Markup Language (PMML) object. A PMML object is an XML-base model to interchange predictive models [53].

First, we have to export the RandomForest model from python environment into a PMML object using the PMMLPipeline class. This was performed using the following snippet:

```
from sklearn.ensemble import RandomForestClassifier

from sklearn2pmml import PMMLPipeline, sklearn2pmml


model_pmml = "rf_model.pmml"

model = RandomForestClassifier()

pipeline = PMMLPipeline([("classifier", model)])

pipeline.fit(X_train, y_train)

sklearn2pmml(pipeline, model_pmml, with_repr = True)
```

Then, the model will be loaded on the Java side using pmml4s library [54], which is a Java library that provides a Java evaluator API for PMML. The values of a given URL's extracted features are given via a Java Map. This was performed using the following snippet:

```
import org.pmml4s.model.Model;
import java.util.*;
public class Model {
        Model  model  =  Model.fromFileMain.class.getClassLoader().getResource
        ("rf_model.pmml").getFile();

        public bool score(Map<String, Double> values) {
                Object[] valuesMap = Arrays.stream(model.inputNames())
                        .map(values::get)
                        .toArray();
                Object[] result = model.predict(valuesMap);
               return (Double) result[0] == 0;
        }
        Map<String, Double> extract_features(url){
                   Map<String, Double> values = Map.of(
                        'FeatureName', value,
                );
              return values;
        }
}
```

6.3 Building Android OS

Since Android is an open source system, anyone can build their own Android system to add interesting features or even change the appearance of interfaces, etc. So, through this chapter we will build our own Android with our URLs detector model.

6.3.1 Downloading AOSP

After many attempts (written in appendices) to install AOSP (Android open-source project). It works on Ubuntu 22.04 as a basic system, first step was downloading all the required packages for working with android source code such as gcc-multilib, g++-multilib, libc6-dev-i386, and many more packages.

The next step was downloading the source code. Before that, we need to download repo tool to get the source code which is a python wrapper script for git. Finally, after 10 hours of waiting the AOSP completed the installation perfectly.

6.3.2 Customizing AOSP - WebView class

After searching a lot for the required modification site, We can add our customization code in the activity class, specifically in startActivity(Intent intent) function. And check if the activity type is ACTION_VIEW which is called for opening URLs. Also, we can add them inside browsers like google chrome and WebView class which is responsible for displaying URLs in this version of Android, as it does not contain any other browser or applications. In WebView class our customization code will locate specifically inside the loadUrl (string url) function.

We used isUrlSafe() function to check if the opened URL is safe, first it call extract_features() function that extract 11 feature from the given URL and returns the features as an array, and then isUrlSafe() check the safety of URL by the score() function that is one of the models functions, and returns true if the URL is benign and false if the URL is malicious [52].

```
public static boolean isUrlSafe(@NonNull String strUri) throws URISyntaxException {
            URI uri = new URI(strUri);


            double features[] = extract_features(uri);
            double[] results = score(features);


            return results[0]==0;
}
```

6.3.3 Building AOSP

First, we must initialize the environment. To dive into the source code.

Command: `source build/envsetup.sh`

Second, choose the target device to install the operating system such as mobile, tablet, or watch. In our case, we chose a mobile device.

Command: `lunch`

Then pick 31 from the menu. Or just enter this command.

Command: `lunch aosp_x86_64-eng`

Finally, start building.

Command: `m-jN` or `make`

N: number of CPU cores allocated to the building process.

After waiting about 14 hours for the building process to complete.

6.3.3.1 Uncustomized AOSP

Android works well and can interact with the imaginary interface effectively without any errors and now we can move to customization process.

6.3.3.2 Customized AOSP without ML model

First write the alert message that will appear to the user when he wants to open any suspicious URL. And with the help of the AlertDialog class, we created a dialog that appears after verifying the security of the URL by using our ML model, with the option Yes to continue opening the URL Figure 25 and the option No to stop opening it Figure 26 using onclick(DialogInterface.onClickListener) Pass it to buttons :

```
//set positive button

.setPositiveButton("Yes",new DialogInterface.OnClickListener(){

   @Override
   public void onClick(DialogInterface dialogInterface, int i){

      //open URL}

   }

})
//set negative button

.setNegativeButton("No", new DialogInterface.OnClickListener() {

   @Override

   public void onClick(DialogInterface dialogInterface, int i) {

          //stop
   }

})
```

6.3.3.3 Customized AOSP with ML model

Then, we create a class in the same webkit package for the model. The model's output controls the appearance of the Alert. Which is return Boolean value about the URL that is user trying to open it. TRUE refers to URL is benign URL and FALSE refers to URL is malicious. Then save all changes and rebuild the source code.

6.3.4 Testing

After we have finished the customization stage, we can test our new system, and we can do it in two ways, the first on the emulator, and the other is on a Physical device.

6.3.4.1 Emulator

The appearance of the AlertDialogue seemed somewhat excellent before adding the model checking process and the response was excellent, but we noticed that when we go to Google to browse and click on some URLs, the AlertDialog does not appear, so it turns out that webView is not responsible for the URLs inside Google, so Google has a method that it uses to open its own URLs.

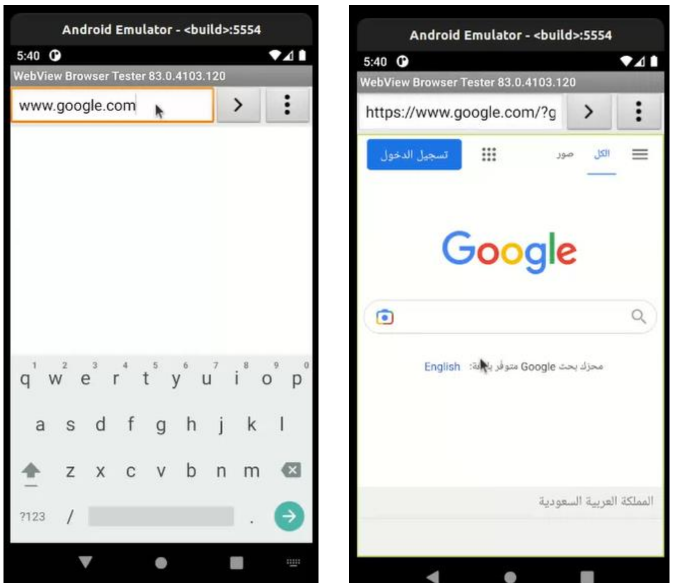
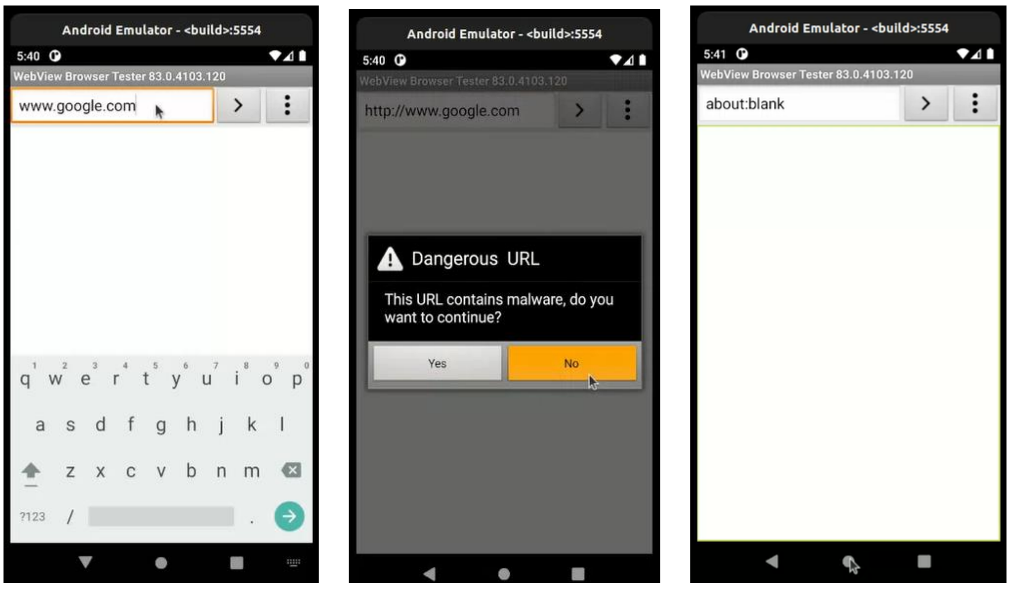

Android Emulator - <build>:5554
WebView Browser Tester 83.0.4103.120
www.google.com
https://www.google.com/?g
تسجيل الدخول
Google
English
المملكة العربية السعودية

Figure 30. When the URL is benign


Android Emulator - <build>:5554
WebView Browser Tester 83.0.4103.120
www.google.com
http://www.google.com
Dangerous URL
Yes
No
about:blank

This URL contains malware, do you want to continue?

Figure 29. When the URL is malicious it won't open.

6.3.4.2 Nokia C2 2nd Edition

An attempt to install System Image on the physical device to test our customized system in real life. To do this we entered the bootloader of the device and then installed our system image. This did not work because the device state is locked, and we couldn’t unlock the state to make changes we tried but time did not help us.

6.4 Summary

In this chapter, we list the lexical features that we will use from the ISCX-URL2016 dataset.in addition, we covered the most popular supervised ML models used in classification. And we discussed the number of classes and samples in the dataset. Also, we talked about the steps of implementation which can be divided into two categories: implementing the ML model and customizing the Android source code. Concerning the first category the algorithm that is used to train the model is RandomForest which had the highest accuracy and recall rate among the other two algorithms, And regarding the customized Android code, we found that the class that would include our customization is the WebView class. Finally, we detailed our experiment. Furthermore, we defined the tools that will be used in training and testing.

Chapter 7 – CONCLUSION

This chapter talks about the limitations and constrains that we faced, and future plans to enhance our idea.

7.1 Limitations of the project

Although our project solves a fundamental problem for many users and makes them feel safe, it has some limitations:

- It is applied to an old version of Android, which is Android 11.
- It only works on WebView, and the version that we worked on does not contain another browser or even applications.
- Our test works only on the emulator, which is a virtual device.
- The size of the model is too large so we couldn’t integrate it into the Android system.
- The supported features in the model are a few.
- Building model in Python then converting it to Java to be compatible with the android customization.

All these points have been taken into consideration and a future plan has been put in place to develop them. And to reduce the limitations of the project.

7.2 Future enhancements/plans

Since this version does not contain applications so we couldn’t do testing on it, Also not even have a modern browser, we can in the future install applications and browsers on this version of Android. Also, we can apply this customization on a newer version of Android, such as Android 13. Finally, integrate our model into the customized Android.

7.3 Summary

Regardless of our project’s usefulness, it has some limitations such as how it can’t be used in newer versions of Android, and how it works only in web view, the size of the model in java, and in future plans we will focus on solving and enhancing these limitations.

APPENDICES

Attempts to customize the AOSP:

- installed ubuntu 18 and 20 on VirtualBox
- installed ubuntu 18 on USB or Hard disk
- Installed ubuntu on amazon cloud (AWS):
    - Installed android version 13 in ubuntu version 20.04 but compilation errors occurred
    - Started from scratch and Installed android version 13 in ubuntu version 20.04, no compilation errors occurred, but since it was very slow we tried to install android version 11 but the cloud storage was filled and the system shut down.
- Started from scratch and installed android version 11 in ubuntu server version 22.04, it worked at first but then a problem in the emulator occurred.
- Remote desktop connection and it didn't work on windows, but it worked on ubuntu using X2Go Client Tool.
- Initially the simulation worked in Android Studio on Windows, it also did in Visual Studio. But when we tried to link the image without customizing the system it didn't work. After looking for another image that was said it would works better with Windows 11, it still didn't work.
- The Ubuntu version that's on the cloud did not work thereby the Oracle Cloud was installed but it is yet rectified.
- The thought was the simulation would work overall better using Ubuntu for the operating system, thus it was used and with the installation of the Android system it all worked, but when compiling to about half of it the terminal stopped. Guessing the problem is caused by the RAM since the device has only 4 RAM, and trying to increase the RAM using a flash memory didn't work. Eventually solved the problem with the Ubuntu that was used as an operating system. Adding 16G RAM to the device, thus totaling to 20G RAM, consequently increasing the number of process unit.
- Installed ubuntu on MacBook UTM, but it didn't work
- Installed VBOX beta version and downloaded ubuntu in it but it was only showing a black screen, not responding, one of the reasons is that the beta version is inadequate.

- Installed multiple different versions of ubuntu on MacBook UTM but all didn’t work and the main reason is that the architecture of MacBook devices is ARM while the architecture of ubuntu is AMD.
- Installed ubuntu version 18 on VBOX, it didn’t work therefore started working on terminator and installed android in it, an error occurred in repo package.
- Installed android version 13.00.16 within ubuntu version 18.04.6, it took over 7 hours for the source code to be downloaded, error on some packages occurred. To try and solve this issue, we changed the android version into 13.00.07 but the same error occurred, ubuntu shut down so this process was repeated from the beginning, and this time we were able to compile the source code but a hardware error occurred where it would give different results each time.
- Repeated compilation of source code multiple times on different architecture types e.g. (32x , 86x).
- Tried installing ubuntu on WSL, and then downloaded android os but it didn’t recognize the hardesk.
- Another version of ubuntu was installed on a different version of UTM and it worked at first but then the storage was filled and the system shut down. This process was repeated again but with the risk of changing the memory size and after 3 hours the source code was installed successfully but once we started lunching the compilation a binary error occurred.
- Tried emulator without compilation but it also didn’t work because it needs compilation.
- Flash system image into samsonge divace and nokia c2 the bootbealder locked.
- After many attempts with nokia it’s worked but the system image is too large.

REFERENCES

[1] Mansi Vijay (2022), Top 5 URL Scanner Tools to Check If a Website is Safe, Review [online], Available from: Top 5 URL Scanner Tools to Check If a Website is Safe (wethegeek.com), (Accessed 12 September 2021).

[2] 1 Zoran Cekerevac, 2 Zdenek Dvorak, 3 Ludmila Prigoda, 4 Petar Cekerevac, Research on 'Hacking, protection and the consequences of hacking' in june 2018 publication at: https://www.researchgate.net/publication/326109904 .

[3] 22 very bad stats on the growth of phishing, ransomware [online], Available from: 22 very bad stats on the growth of phishing, ransomware | VentureBeat (Accessed 12 September 2022).

[4] Jansson, K.; von Solms, R. (2019-11-09). "Phishing for phishing awareness". Behaviour & Information Technology. 32 (6): 584–593.

[5] www.javatpoint.com. 2022. Kernel Vs. Operating System - javatpoint. [online] Available at: <https://www.javatpoint.com/kernel-vs-operating-system> [Accessed 11 October 2022].

[6] Baird, S., 2022. What is a Process?. [online] ProcessModel. Available at: <https://www.processmodel.com/blog/what-is-a-process/> [Accessed 7 October 2022].

[7] Amazon Web Services, Inc. 2022. What is an API? - API Beginner's Guide - AWS. [online] Available at: <https://aws.amazon.com/what-is/api/> [Accessed 2 October 2022].

[8] U Farooq - Lahore, 'Android Operating System Architecture', Virtual University of Pakistan, 2018.

[9] V. Humeniuk, 'Android Architecture Comparison: MVP vs. VIPER', Dissertation, 2019.

[10] Binils.com. 2022. [online] Available at: <https://www.binils.com/wp-content/uploads/2021/11/CS8493-UNIT-5-Android-Architecture.pdf> [Accessed 11 October 2022].

[11] Vidas, Timothy, Daniel Votipka, and Nicolas Christin. A Survey of Current Android Attacks. n.d. https://www.usenix.org/legacy/events/woot11/tech/final_files/Vidas.pdf (accessed oct 2022). [8] Android. Secure an Android Device. Sep 13, 2022. https://source.android.com/docs/security/overview.

[12] Android. Application security. Sep 2022. https://source.android.com/docs/security/overview/app-security.

[13] Android. Application Sandbox. Oct 2022. https://source.android.com/docs/security/app-sandbox.

[14] Android. Application Signing. Oct 11, 2022. https://source.android.com/docs/security/features/apksigning.

[15] Android. Kernal security. Oct 4, 2022. https://source.android.com/docs/security/overview/kernel-security.

[16] J. Scarpati and J. Burke, "URL (Uniform Resource Locator)," TechTarget, Sep 2021. https://www.techtarget.com/searchnetworking/definition/URL.

[17] "Components of a URL," geeksforgeeks, 29 jun 2021. https://www.geeksforgeeks.org/components-of-a-url/.

[18] "Malicious urls | ClearNetwork, Inc." [Online]. Available: https://www.clearnetwork.com/malicious-urls/. [Accessed: 28-Oct-2022].

[19] Savvy Security, "What is a malicious url? (and how you can avoid them)," Savvy Security, 22-Apr-2021. [Online]. Available: https://cheapsslsecurity.com/blog/what-is-a-malicious-url/. [1]

[20] "What is a malicious URL?," gatefy, 18-Mar-2021. [Online]. Available: https://gatefy.com/blog/what-malicious-url/.

[21] C. Do Xuan, H. D. Nguyen and V. N. Tisenko, "Malicious URL Detection based on Machine Learning," International Journal of Advanced Computer Science and Applications, vol. 11, no. Science and Information (SAI) Organization Limited, 2020.

[22] "What is an antivirus product? do I need one?," NCSC, 21-Jan-2019. [Online]. Available: https://www.ncsc.gov.uk/guidance/what-is-an-antivirus-product.

[23] Virustotal. [Online]. Available: https://www.virustotal.com/gui/home/url. [Accessed: 28-Oct-2022].

[24] "Cybersecurity delivered: Sophos Security Solutions," SOPHOS. [Online]. Available: https://www.sophos.com/en-us/products/mobile-control/intercept-x.

[25] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL Detection using Machine Learning: A Survey." Cornell University, Ithaca, 21-Aug-2019.

[26] G. Wejinya and S. Bhatia, "Machine Learning for Malicious URL Detection," in Advances in Intelligent Systems and Computing, vol. 1270, Singapore: Springer Nature, 2020, pp. 463–472.

[27] M. Ferreira, Malicious URL detection using machine learning algorithms, in Digital Privacy and Security Conference (2019), p. 114

[28] - Johnson, C., Khadka, B., Basnet, R. B., & Doleck, T. (2020). Towards Detecting and Classifying Malicious URLs Using Deep Learning. J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl., 11(4), 31-48.

[29] – Divya, K., Anupriya, A. B., Nidi, M., & Aditya, J. (2019). Machine Learning Based Malicious URL Detection. International Journal of Engineering and Advanced Technology, 8(4), 1-5.

[30] – Bhanuprakash, S., Keerthana, M. G., & Murthy, S. V. B. (2022). DETECTING MALICIOUS FACEBOOK APPLICATIONS.

[31] - Alsaedi, M., Ghaleb, F. A., Saeed, F., Ahmad, J., & Alasli, M. (2022). Cyber Threat Intelligence-Based Malicious URL Detection Model Using Ensemble Learning. Sensors, 22(9), 3373.

[32] Tung, S. P., Wong, K. Y., Kuzminykh, I., Bakhshi, T., & Ghita, B. (2021). Using a Machine Learning Model for Malicious URL Type Detection. In Internet of Things, Smart Spaces, and Next Generation Networks and Systems (pp. 493-505). Springer, Cham.

[33] Zeng, Y. (2018). Malicious urls and attachments detection on lexical-based features using machine learning techniques.

[34] A. Cuzzocrea, F. Martinelli and F. Mercaldo , "A machine-learning framework for supporting intelligent web-phishing detection and analysis," in IDEAS '19: Proceedings of the 23rd International Database Applications & Engineering Symposium, New York, 2019.

[35] A. Joshi, L. Lloyd, P. Westin and S. Seethapathy, "arXiv, Cornell University," 14 10 2019. [Online]. Available: https://arxiv.org/abs/1910.06277. [Accessed 10 10 2022].

[36] A. B. Ramesh and S. K. Sab, “URL Classification Using whois and Lexical Features,” p. 4.

[37] M. H. Alshira'H and M. Al-Fawa'reh, "Detecting Phishing URLs using Machine Learning & Lexical Feature-based Analysis," *International Journal of Advanced Trends in Computer Science and Engineering,* vol. 9, no. 4, pp. 5828-5837, 2020.

[38] G. Wejinya and S. Bhatia, “Machine Learning for Malicious URL Detection,” in *Advances in Intelligent Systems and Computing*, vol. 1270, Singapore: Springer Nature, 2020, pp. 463–472.

[39] Ahammad, S. H., Kale, S. D., Upadhye, G. D., Pande, S. D., Babu, E. V., Dhumane, A. V., & Bahadur, M. D. K. J. (2022). Phishing URL detection using machine learning methods. Advances in Engineering Software, 173, 103288.

[40] “Wot Mobile Security Protection - Apps on Google Play,” Google. [Online]. Available: https://play.google.com/store/apps/details?id=com.wot.security&amp;hl=en&amp;gl=US. [Accessed: 29-Oct-2022].

[41] A. McFarland, “10 ‘Best’ antivirus apps for Android (October 2022),” Unite.AI, 01-Oct-2022. [Online]. Available: https://www.unite.ai/10-best-antivirus-apps-for-android/. [Accessed: 29-Oct-2022].

[42] “Link protector: URL security - apps on Google Play,” Google. [Online]. Available: https://play.google.com/store/apps/details?id=com.swarajyadev.linkprotector&amp;hl=en&amp;gl=US. [Accessed: 29-Oct-2022].

[43] “Scan URL - link Checker - apps on Google Play,” Google. [Online]. Available: https://play.google.com/store/apps/details?id=com.seotools.scanurl. [Accessed: 29-Oct-2022].

[44] abhisheksaxena1998, “ABHISHEKSAXENA1998/android-application-malicious-URL-detector: Android-Application-Malicous-URL-detector,” GitHub. [Online]. Available: https://github.com/abhisheksaxena1998/Android-Application-Malicious-URL-Detector. [Accessed: 29-Oct-2022].

[45] 20 experts share secrets for balancing agile and Waterfall. (n.d.). Retrieved October 23, 2022, from https://business.adobe.com/resources/ebooks/20-experts-share-secrets-for-balancing-agile-and-waterfall.html.

[46] Bhise, S., Gadekar, S., Gaur, A. S., Bepari, S., & Deepmala Kale, D. S. A. (2021). Breast cancer detection using machine learning techniques. International Journal of Engineering Research & Technology (IJERT), 10(7).

[47] Mahesh, B. (2020). Machine learning algorithms-a review. International Journal of Science and Research (IJSR). [Internet], 9, 381-386.

[48] Wejinya, G., & Bhatia, S. (2021). Machine learning for malicious url detection. In ICT Systems and Sustainability (pp. 463-472). Springer, Singapore.

[49] Canadian Institute for Cybersecurity: URL dataset (ISCX-URL-2016).

[50] “The layers of the android security model | proandroiddev - medium.” [Online]. Available: https://proandroiddev.com/the-layers-of-the-android-security-model-90f471015ae6. [Accessed: 30-Oct-2022].

[51] BayesWitnesses Bayeswitnesses/m2cgen: Transform ML models into a native code (Java, C, python, go, JavaScript, visual basic, C#, R, PowerShell, PHP, Dart, Haskell, ruby, F#, rust) with Zero dependencies, GitHub. Available at (Accessed: February 9, 2023).

[52] Bajahlan, J. (no date) *ProJoman/java-features-extraction: Extracting 11 features from urls dataset by Java*, *GitHub*. Available at: https://github.com/ProJoman/java-features-extraction (Accessed: February 9, 2023).

[53] Grossman, R., Bailey, S., Ramu, A., Malhi, B., Hallstrom, P., Pulleyn, I., & Qin, X. (1999). The management and mining of multiple predictive models using the predictive modeling markup language. Information and Software Technology, 41(9), 589-595.

[54] Marache, M. and Junod, M. (2023) *PMML scoring library for Scala (PMML4S)*, AutoDeploy.AI. Online: https://github.com/autodeployai/pmml4s.