MALWARE ATTACK DETECTION (M.A.D)

Project Report Submitted

In Partial Fulfilment of the Requirements for

the Degree of

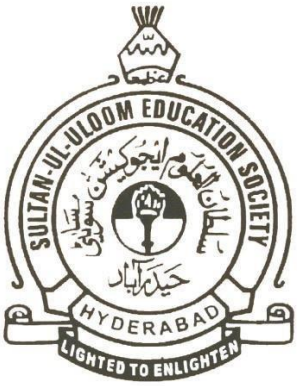
BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

Submitted By

Suhaib Ahmed Sayeed (1604-17-737-032)
Shajee Ahmed Musharaf (1604-17-737-096)
Mohd Talha Haseeb (1604-17-737-104)



INFORMATION TECHNOLOGY DEPARTMENT
MUFFAKHAM JAH COLLEGE OF ENGINEERING& TECHNOLOGY

(Affiliated to Osmania University)
Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad-34 2019-20

CERTIFICATE

It is certified that the work contained in the project report titled "Malware Attack Detection (M.A.D)," by

Mr. Suhaib Ahmed Sayeed	**(1604-17-737-032)**
Mr. Shajee Ahmed Musharaf	**(1604-17-737-096)**
Mr. Mohd Talha Haseeb	**(1604-17-737-104)**

has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree"

Project Supervisor

Mr. Himayathullah Sharief
Information Technology Dept.
MJ College of Engineering & Tech.
Hyderabad – 500034

Signature of Head

Dr.MOUSMI AJAYCHAURASIA
Information Technology Dept.
MJ College of Engineering & Tech.
Hyderabad - 500034

External Examiner

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express our heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mr. Himayatullah Sharief,** Professor, Information Technology Department for his valuable suggestions and guidance during the execution of this project work.

We would like to thank **Dr. MOUSMI AJAY CHAURASIA**, Head of Information Technology Department, for her moral support throughout the period of my study in MJCET.

We are highly indebted to, Principal MJCET for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the Teaching & Non- teaching staff of Information Technology Department for their co-operation

Finally, we express our sincere thanks to **Prof. Basheer Ahmed**, Advisor cum Director, MJCET, for his continuous care. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

Mr. Suhaib Ahmed Sayeed 1604-17-737-032
Mr. Shajee Ahmed Musharaf 1604-17-737-096
Mr. Mohd Talha Haseeb 1604-17-737-104

ABSTRACT

The World Wide Web supports a wide range of criminal activities such as spam-advertised e-commerce, financial fraud and malware dissemination. Although the precise motivations behind these schemes may differ, the common denominator lies in the fact that unsuspecting users visit their sites. These visits can be driven by email, web search results or links from other web pages. In all cases, however, the user is required to take some action, such as clicking on a desired Uniform Resource Locator.

In order to identify these malicious sites, the web security community has developed blacklisting services. These blacklists are in turn constructed by an array of techniques including manual reporting social engineering, phishing, pharming and web crawlers combined with site analysis heuristics. Inevitably, many malicious sites are not blacklisted either because they are too recent or were never or incorrectly evaluated.

As a results, malicious URL detection is of great interest nowadays. There have been several scientific studies showing a number of methods to detect malicious URLs based on machine learning and deep learning techniques. In this project, we propose a malicious URL detection method using machine learning techniques based on our proposed URL behaviours and attributes. Moreover, Big Data technology is also exploited to improve the capability of detection malicious URLs based on abnormal behaviours. In short, the proposed detection system consists of a new set of URLs features and behaviours, a machine learning algorithm, and a Big Data technology. The experimental results show that the proposed URL attributes and behaviour can help improve the ability to detect malicious URL significantly. This is suggested that the proposed system may be considered as an optimized and friendly used solution for malicious URL detection.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Overview

Uniform Resource Locator (URL) is used to refer to resources on the Internet. The characteristics and two basic components of the URL are: protocol identifier, which indicates what protocol to use, and resource name, which specifies the IP address or the domain name where the resource is located. It can be seen that each URL has a specific structure and format. Attackers often try to change one or more components of the URL's structure to deceive users for spreading their malicious URL. Malicious URLs are known as links that adversely affect users. These URLs will redirect users to resources or pages on which attackers can execute codes on users' computers, redirect users to unwanted sites, malicious website, or other phishing site, or malware download. Malicious URLs can also be hidden in download links that are deemed safe and can spread quickly through file and message sharing in shared networks. Some attack techniques that use malicious URLs include : Drive-by Download, Phishing and Social Engineering, and Spam

According to statistics , in 2019, the attacks using spreading malicious URL technique are ranked first among the 10 most common attack techniques. Especially according to this statistic, the three main URL spreading techniques, which are malicious URLs, botnet URLs, and phishing URLs, increase in number of attacks as well as danger level.

From the statistics of the increase in the number of malicious URL distributions over the consecutive years, it is clear that there is a need to study and apply techniques or methods to detect and prevent these malicious URLs.

Regarding the problem of detecting malicious URLs, there are two main trends at present as malicious URL detection based on signs or sets of rules, and malicious URL detection based on behaviour analysis techniques . The method of detecting malicious URLs based on a set of markers or rules can quickly and accurately detect malicious URLs. However, this method is not capable of detecting new malicious URLs that are not in the set of predefined signs or rules. The method of detecting malicious URLs based on behaviour analysis techniques adopt machine learning or deep learning algorithms to classify URLs based on their behaviours. In this project, machine learning

algorithms are utilized to classify URLs based on their attributes. The project also includes a new URL attribute extraction method. In our project, machine learning algorithms are used to classify URLs based on the features and behaviours of URLs. The features are extracted from static and dynamic behaviours of URLs. Those newly proposed features are the main contribution of the project. Machine learning algorithms are a part of the whole malicious URL detection system. Two supervised machine learning algorithms are used, Support vector machine (SVM) and Random forest (RF).

1.2 Problem Statement

To develop an automatic malware detection tool that ensures data security by reducing the huge threat caused by the malware data present implicitly in web applications.

1.3 Objectives

1. Employing various data mining concepts
2. Feature extraction of URLs
3. Use of machine learning algorithms.

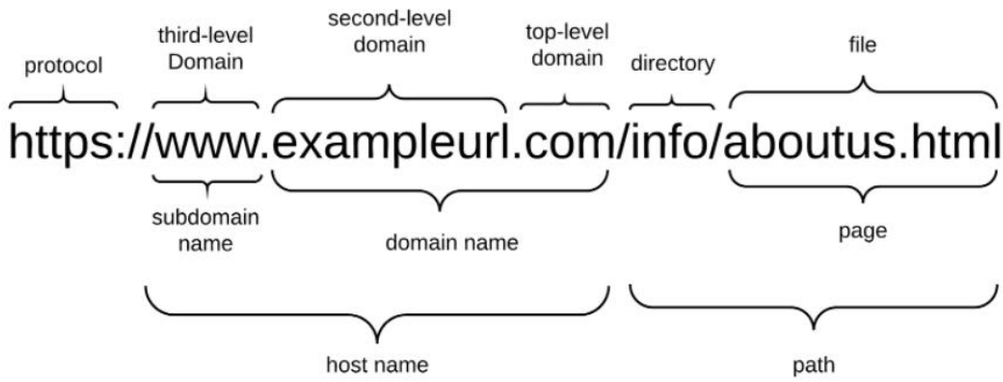


Figure.1.1.Sample features of URL for extraction

1.4 Types of Features to be extracted

1.4.1. Having IP address: The nature of IP addresses is based on the URL—if the IP address exists in the URL instead of the domain name, this typically means there has been an attempt to hijack or steal personal information; otherwise, the webpage would be considered legitimate.

1.4.2. Port: In this case, if a port is compromised, all hosted IPs are affected. If the IP address is affected, then only specific webpages associated with that IP are affected, while the port remains safe. Malicious attacks on port are less common compared to IP address manipulation.

1.4.3. Request URL: Based on the previous two analyses, it may be concluded that malware occurrence through ports is relatively infrequent while 'request URL' has a strong influence on malware behaviour.

1.4.4. Google index: Based on the previous three analyses, it is clear that the number of occurrences of malware attacks through Google index and through ports is nearly similar

1.4.5. Web traffic: The nature of this feature is based on the number of visitors to the webpage. In the phishing dataset, the number of webpages with malicious traffic was less than the number of legitimate webpages. The interesting finding in this feature was that suspicious never indicates whether it is legitimate or phishy.

.1.4.6. Abnormal URL: The nature of this feature is based on the identity of URL. If a URL included the host name, it was considered legitimate; otherwise, it was considered phishy.

1.4.7. Pop-up window: The function of pop-up windows in webpages is to ask users for some credentials. In the current data, 8918 webpages were found that did not use pop-up windows, which classified them as legitimate, whereas 2137 were phishy. Some pop-ups are based on adware, which is a next-generation malware, meaning that information regarding the suspiciousness of this feature was not present in this dataset.

1.4.8. Links pointing to page: This feature refers to links pointing to a specific URL (i.e., page or subpage).

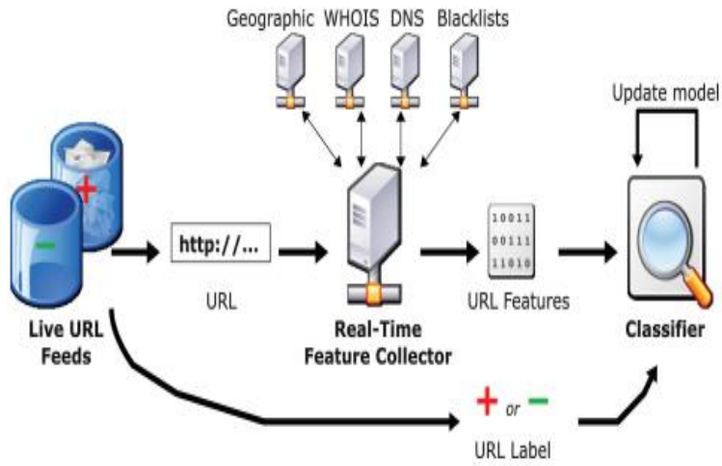


Figure.1.4.Real time feature collector to update existing model

1.5 Organization of Report

This rest of the report is laid out as follows. Chapter 2 Literature Survey presents an extensive review on the existing methods and technology applied along with the parameters involved. Chapter 3 System Analysis is an overview of the malware attack detection process and the proposed solution of our project along with all the technical specifications. A general framework of the proposed method is presented Chapter 4 i.e. System Design which is followed by conclusion, future scope and references section, also including appendix 1 and 2 at the end of the report.

2. LITERATURE SURVEY

2.1 Related research projects

2.1.1 Signature based Malicious URL Detection

Studies on malicious URL detection using the signature sets had been investigated and applied long time ago. Most of these studies often use lists of known malicious URLs. Whenever a new URL is accessed, a database query is executed. If the URL is blacklisted, it is considered as malicious, and then, a warning will be generated; otherwise URLs will be considered as safe. The main disadvantage of this approach is that it will be very difficult to detect new malicious URLs that are not in the given list.

2.1.2 Phishing URL Detection Via CNN And Attention-Based Hierarchical RNN

- Proposed Phishing Net (Deep Learning Approach)
- Used CNN for character-level feature extraction
- Used Attention-based hierarchical RNN for word-level feature extraction
- Fused character and word-level features via CNN
- Took longer execution time while fusing CNN and RNN

2.1.3 Phishing URL Detection Via Capsule-Based Neural Network

- Proposed Capsule-based neural network
- Primary capsule layer: extracted accurate features from shallow features generated by former convolution layer and utilized batch normalization
- Classification capsule layer: used dynamic routing algorithm and squashing function and averaged outputs from all branches
- Lower true positive rate

2.1.4 Malicious Domain Name Detection Based On Extreme Machine Learning

- Proposed machine learning based methodology using Extreme Learning Machine (ELM) for malicious domain detection
- Used Single-hidden-Layer-Feed forward networks (SLFNs) and moduled detection problem as SLFN
- Detection rate and accuracy dropped if no of nodes are larger than 1000

2.1.5 URLNet: Learning A URL Representation With Deep Learning For Malicious URL Detection

- Proposed end-to-end deep learning framework, URLNet
- Learn nonlinear embedding directly from URLs
- Applied CNN for both character- and word-level embedding

2.1.6 Acquire, Adopt, And Anticipate: Continuous Learning To Block Malicious Domains Proposed end-to-end deep learning framework, URLNet

- Proposed automated learning system
- Develops deep learning model
- Publishes unreported malicious domains
- Periodically updates detection models

2.1.7 Malicious URL Detection Tools

- URL Void: URL Void is a URL checking program using multiple engines and blacklists of domains. Some examples of URL Void are Google SafeBrowsing, Norton SafeWeb and MyWOT. The advantage of the Void URL tool is its compatibility with many different browsers as well as it can support many other testing services. The main disadvantage of the Void URL tool is that the malicious URL detection process relies heavily on a given set of signatures.

- UnMask Parasites: Unmask Parasites is a URL testing tool by downloading provided links, parsing Hypertext Markup Language (HTML) codes, especially external links, iframes and JavaScript. The advantage of this tool is that it can detect iframe fast and accurately. However, this tool is only useful if the user has suspected something strange happening on their sites.

- Dr.Web Anti-Virus Link Checker: Dr.Web Anti-Virus Link Checker is an add-on for Chrome, Firefox, Opera, and IE to automatically find and scan malicious content on a download link on all social networking links such as Facebook, Vk.com, Google+.

- Comodo Site Inspector: This is a malware and security hole detection tool. This helps users check URLs or enables webmasters to set up daily checks by downloading all the specified sites. and run them in a sandbox browser environment.

- Some other tools: Among aforementioned typical tools, there are some other URL checking tools, such as UnShorten.it, VirusTotal, Norton Safe Web, SiteAdvisor (by McAfee), Sucuri, Browser Defender, Online Link Scan, and Google Safe Browsing Diagnostic.

2.1.8 Classification algorithms

a) Random forest: Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.

2.1.9 Ensemble methods

The goal of ensemble methods is to combine the prediction of several base estimators built with a given learning algorithm in order to improve generalizability/ robustness over a single estimator. Two families of ensemble methods re usually distinguished: one is the averaging method and the other is boosting method.

2.1.9.1 Controlling the tree size: The size of the regression tree base learners defines the level of variable interactions that can be captured by the gradient boosting model. In general, a tree of depth h can capture interactions of order h. There are two ways in which the size of the individual regression trees can be controlled.

2.1.9.2 Regression: It supports a number of different loss functions for regression, the default loss function for regression is at least squares.

2.1.9.3 Comparison between various works: We compare our scheme with related work in terms of features, computation, and space requirement. We exclude the cost and space for those certificates verification in this comparison, as it may vary in different scenarios. In the earlier inventions of this project, the use of stacking classifier was not introduces whereas in this project the stacking classifier is employed here which performs the computation, by comparing all the algorithms and gives its result to meta classifier. The Meta Classifier gives the best algorithm depending on the high accuracy of such algorithm. Hence higher accuracy is achieved here.

2.2 Technology Used

2.2.1 Random Forest

Random Forest tries to build multiple CART models with different samples and different initial variables. For instance, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction. The RF classification algorithm is used in two phases. First, the RF algorithm extracts subsamples from the original samples using the bootstrap re-sampling method and creates decision trees for each sample. Second, the algorithm classifies the decision trees and implements a simple vote, with the largest vote of the classification as the final result of the prediction.

The RF algorithm always includes two steps as follows:

(1) Select the training set. Use the bootstrap random sampling method to retrieve K training sets from the original dataset (M properties), with the size of each training set the same as that of the original training set.

(2) Build the RF model. Create a classification regression tree for each of the bootstrap training sets to produce K decision trees to form a “forest”; these trees are not pruned. Looking at the growth of each tree, this approach does not choose the best features as internal nodes for branches but rather the branching process is a random selection of all features.

2.2.2 Classifier Evaluation Index

The common evaluation indices for the prediction model’s performance are accuracy (ACC), recall, precision (PPV), and the area under the curve (AUC). To calculate these indices, the confusion matrix is used. In the matrix, the columns represent the prediction categories and the sum of the value in the column is the data observations in the category. In addition,

the rows in the matrix represent the actual categories and the sum of the values in the rows represents the data observations in that category. In this study, our focus is on whether or not there is malware, which is a binary classification. Safe is set as the positive category and malicious set as the negative category Recall denotes the true positive rate (TPR) and the equation is

- $Recall = TPR = TP/(TP + FN)$ (1)

FPR denotes the false positive rate and the equation is

- $FPR = FP/(FP + TN)$ (2)

Precision denotes the positive predictive value (PPV) and the equation is

- $PPV = TP/(TP + FP)$ (3)

ACC denotes accuracy and the equation is

- $ACC = (TP + TN)/(TP + FP + FN+ TN)$ (4)

In this study, we pay attention to the small classes (categorization features as turnover) in the problem of unbalanced classification. The main goal is to avoid misdiagnosis and minimize misdetection. Therefore, recall, the F-measure, the AUC, and the overall ACC will be used to evaluate the performance of the classification algorithm.

2.2.3 Anaconda

It is a free and open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing. The distribution makes package management and deployment simple and easy. Matplotlib and lots of other useful (data) science tools form part of the distribution.

Figure.2.2.3 Anaconda Platform

2.2.4 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Figure.2.2.4 Jupyter Notebook

3. SYSTEM ANALYSIS

3.1 Problem with Existing System

According to statistics presented in 2019, the attacks using spreading malicious URL technique are ranked first among the 10 most common attack techniques. Especially, according to this statistic, the three main URL spreading techniques, which are malicious URLs, botnet URLs, and phishing URLs, increase in number of attacks as well as danger level.

From the statistics of the increase in the number of malicious URL distributions over the consecutive years, it is clear that there is a need to study and apply techniques or methods to detect and prevent these malicious URLs.

Regarding the problem of detecting malicious URLs, there are two main trends at present as malicious URL detection based on signs or sets of rules, and malicious URL detection based on behaviour analysis techniques. The method of detecting malicious URLs based on a set of markers or rules can quickly and accurately detect malicious URLs. However, this method is not capable of detecting new malicious URLs that are not in the set of predefined signs or rules. The method of detecting malicious URLs based on behaviour analysis techniques adopt machine learning or deep learning algorithms to classify URLs based on their behaviours. In this project, machine learning algorithms are utilized to classify URLs based on their attributes. The project also includes a new URL attribute extraction method.

In our project, machine learning algorithms are used to classify URLs based on the features and behaviours of URLs. The features are extracted from static and dynamic behaviours of URLs and are new to the technological domain. Those newly proposed features are the main contribution of our project. Machine learning algorithms are a part of the whole malicious URL detection system. Two supervised machine learning algorithms are used, Support vector machine (SVM) and Random forest (RF).

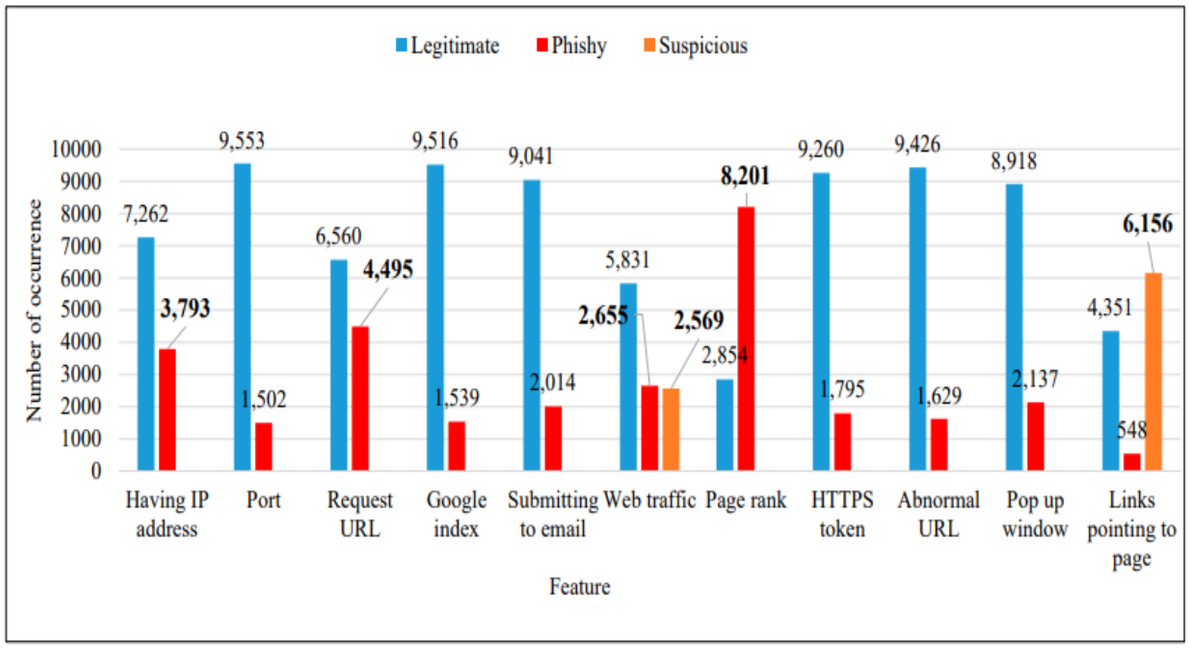


Figure.3.1 Number of occurrences of URL features

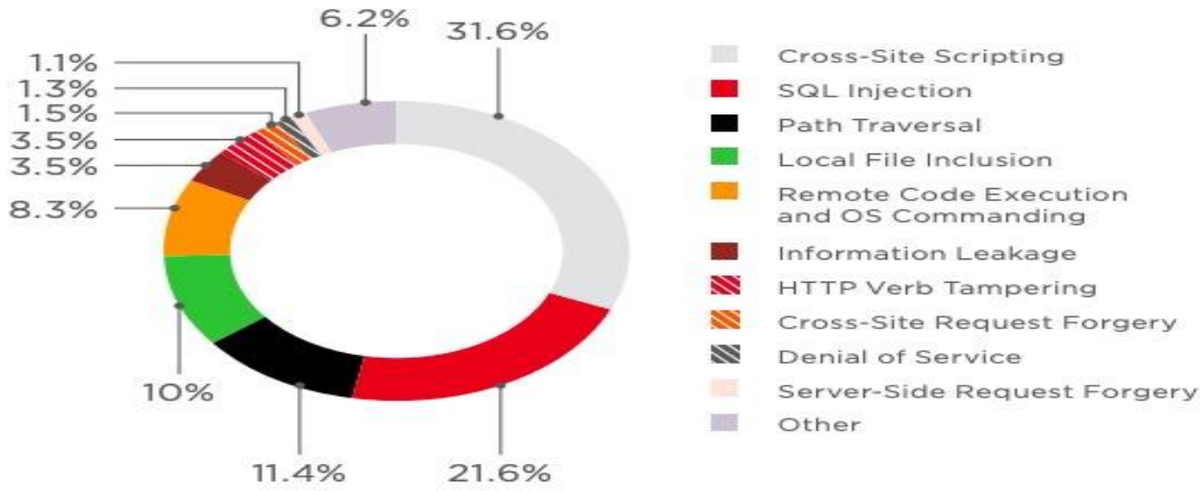


Figure.3.1.2 Number of Attacks on Websites

3.2 Proposed Work

The proposed system has the following:

3.2.1 Data set

The experimental dataset for malicious URL detection model includes: 470.000 URLs collected from different resources, of which about 70.000 URLs are malicious and 400.000 URLs are safe. All these URLs are checked by Virus Total tool to verify the labels of each URL. The complete dataset is stored using CSV format. Each URL sample has a label "bad" for malicious and "good" for safe. Details of the data are as follows:

- Phishtank: Phishtank is a service Website dedicated for sharing phishing URLs. Suspicious URLs can be sent to Phishtank for verification. The data in Phishtank is updated hourly.
- URLhaus: URLhaus is a project from abuse.ch aiming at sharing malicious URLs being used for malicious software distribution.
- Alexa: Is a database ranking all websites according to their usefulness.
- Malicious_n_Non-Malicious URL: is a data source with more than 400,000 labeled URL. In this database, 82% of all URLs are safe, while remaining 18% of URLs are malicious.

3.2.2 Data pre-processing

Preprocessing the data is an essential and significant step to improve the machine learning results. A valid and clean data should be used as an input for the algorithms. In this research, we used a tableau, which is a software developed for data visualization . The processing of the data was done by deleting with the noisy data, which causes problems in the performance of the algorithms. In addition, the results are improved by deleting the content length column for containing a lot of null

Feature construction: aims to create additional features with a better discriminative ability than the initial feature set. This can bring significant improvement to machine learning algorithms. Features can be constructed manually, or by using data mining methods such as sequence analysis, association mining, and frequent-episode mining.

3.2.3 Test dataset and training dataset

Separating data into test datasets and training datasets is an important part of evaluating data mining models. By this separation of total data set into two data sets we can minimize the effects of data inconsistency and better understand the characteristics of the model. The test data set contains all the required data for data prediction and training data set contains all irrelevant data. The dataset of both safe and malicious URLs mentioned above is divided into 2 subsets. About 80% of the dataset, 470.000 URLs (400.000 safe URLs, 70.000 malicious URL), is used for training, and about 20% of the dataset, about 10.000 URLs (5.000 malicious URLs, 5.000 safe URLs), is used for testing. The experiment is repeated many times with both SVM and RF algorithm.

Dataset	Algorithm and parameters	Accuracy (%)	Precision (%)	Recall (%)	Training time (s)	Testing time (s)
10.000 URLs	SVM (100 iterations)	93.39	94.67	92.51	2.32	0.01
	SVM (10 iterations)	93.35	94.84	92.71	3.11	0.01
	RF (10 trees)	99.10	98.43	97.45	2.78	0.01
	RF (100 trees)	99.77	98.75	97.85	3.34	0.01
470.000 URLs	SVM (100 iterations)	90.70	93.43	88.45	272.97	2.12
	SVM (10 iterations)	91.07	93.75	88.85	280.33	2.31
	RF (10 trees)	95.45	90.21	95.12	372.97	2.02
	RF (100 trees)	96.28	91.44	94.42	480.33	2.30

Table.3.2.3.1. Training of the Data Set

	Predicted Safe URL	Predicted malicious URL
Real safe URL (107)	96	11
Real malicious URL (1180	9	109

Table 3.2.3.2 Testing of the Data Set

3.2.4 Data classification techniques:

Data classification is the process of organizing data into categories for its most effective and efficient use. Data classification techniques are :

3.2.4.1 Decision Tree: It is tree structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branched notes the outcome of a test, and each leaf node holds a class label.

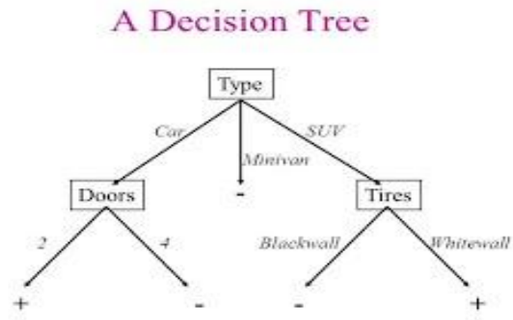


Figure 3.2.4.1 Working of Decision Tree

3.2.4.2 Support Vector Machine: In machine learning, support-vector machines (SVMs also support-vector networks) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

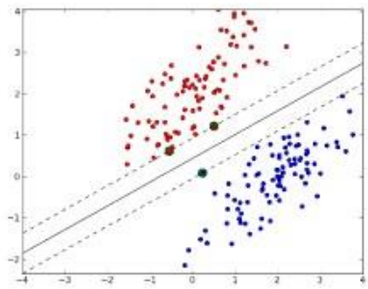


Figure 3.2.4.2. Working of Support Vector Machine

3.2.4.3 Random Forest: Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e.

multiple decision trees, resulting in a forest of trees, hence the name "Random Forest".

The random forest algorithm can be used for both regression and classification tasks.

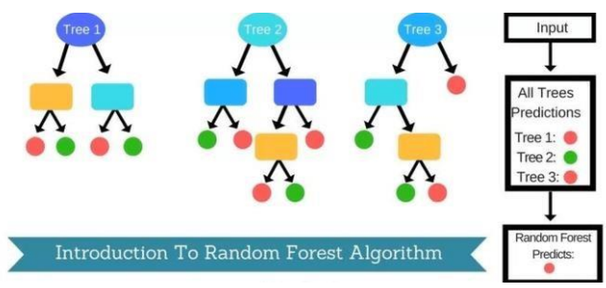


Figure 3.2.4.3. Working of Random Forest

3.3 Feasibility Study

Feasibility analysis is the process of confirming that a strategy, plan or design is possible and makes sense. This can be used to validate assumptions, constraints, decisions, approaches and business cases. For this project, we will be discussing the three major types of feasibility that are technical feasibility, operational feasibility and economic feasibility.

3.3.1 Technical Feasibility

A Technical feasibility study assesses the details of how you intend to deliver a product or service to customers. It is the process of validating the technology assumptions, architecture and design of a product or project. The proposed system consists of free-to-use software and it is platform independent. It can be used with different software versions by only doing minor changes to the code and has some different execution steps involved. This project is mainly used by many companies to reduce their financial costs by using retention schemes for the leaving employees.

3.3.2 Economical Feasibility

Economic feasibility is the cost and logistical outlook for a business project or endeavour. It is a kind of cost-benefit analysis of the examined project, which assesses whether it is possible to implement it or not. This project uses the Python libraries like Pandas, NumPy, SciPy, matplotlib, sklearn, etc using Anaconda and it also uses the Flask Environment. Python is a free, open-source programming language that is available for everyone to use. We have used the Jupyter notebook for the graphical representation of the algorithms and to find the accuracy of each individually. We then used Spyder notebook to implement the flask environment and the front end was executed through the web browser by running the flask on Anaconda PowerShell.
From this, we can say that the proposed system is economically feasible.

3.3.3 Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems. This system operates based on the various features of the URL and using the best algorithms accurately finds which of the URL will contain malware or not based on features like google index, host rank, port etc. This needs proper and valid data for the URL, with that kind of data it is highly feasible. Although a few precautions have to be kept in mind for each URL data.

3.4 Software Requirement Specification

3.4.1 Introduction

3.4.1.1 Purpose

• We propose several groups of novel, highly discriminative features that enable our method to deliver a superior performance and capability on both detection and threat-type identification of malicious URLs of main attack types including spamming, phishing, and malware infection. Our method provides a much larger coverage than existing methods while maintaining a high accuracy.

• To the best of our knowledge, this is the first study on classifying multiple types of malicious URLs, known as a multi-label classification problem, in a systematic way. Multi-label classification is much harder than binary detection of malicious URLs since multi-label learning has to deal with the ambiguity that an entity may belong to several classes

3.4.1.2 Scope

The main scope of this project is to analyze the effectiveness of chosen machine learning classification technologies within the problem of malicious URL detection. Specifically, we are interested in results using only URL address itself without the need to download potentially risky content of the page. We consider the problem of detecting malicious URLs as a binary classification problem, i.e., URL can be benign or malicious. Determining the type of malicious URL may be possible, but it is usually not the main goal of detection algorithms. In the beginning, we analyzed and selected the set of features that may have a positive impact on the results. In the next step, we extracted and preprocessed these features.

We decided to use Support Vector Machine, as a representative for batch learning and Random forests that are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.

3.4.2 Overall Description

3.4.2.1 Product Perspective

This product is mainly introduced for the daily users to detect the malwares in the websites before it even happens to reduce the financial cost of the company. It uses the main features of all the website stored in the record and these can be evaluated using the machine learning algorithms and the result can be estimated immediately on just entering the URL.. Through this we can detect any type of possible Cyber Attack by making necessary retention plans for Web Applications.

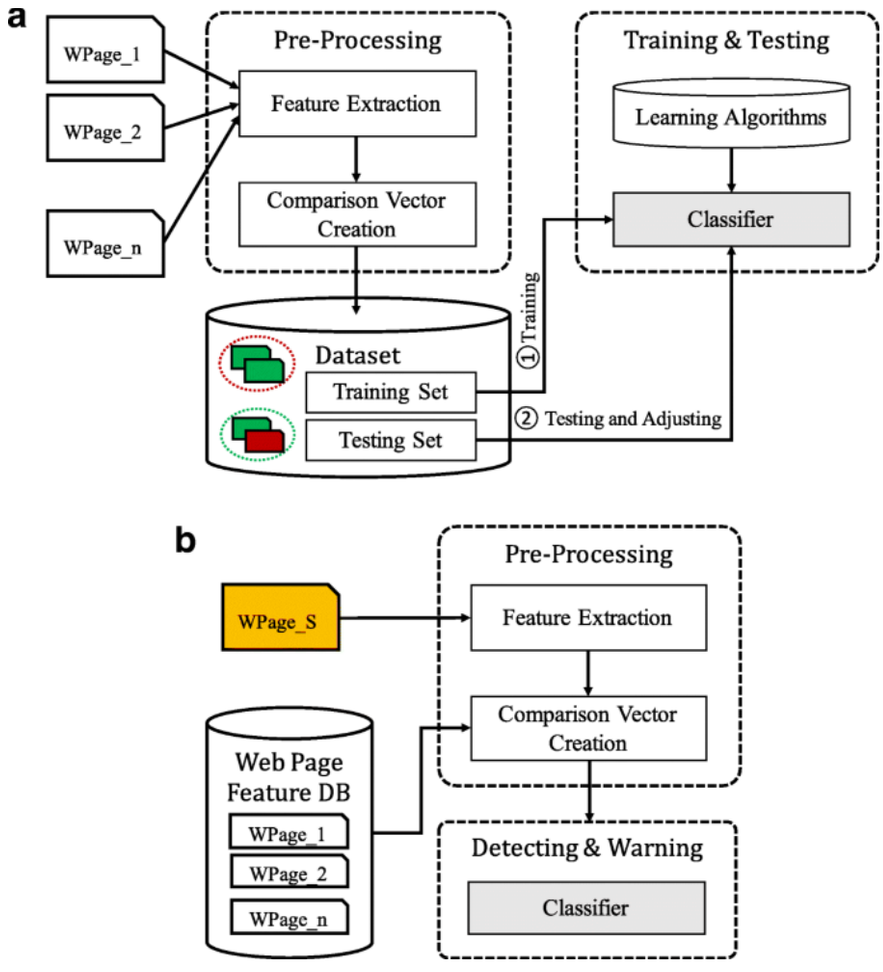


Figure. 3.4.2.1 Block Diagram of the product

3.4.2.2 Product Functions

The product's goal is to detect malicious URLs and alert the user to prepare Incident Response for that particular Web Application.

The functions of the products are as follows:

1. To take the input from the user to check about particular website
2. Use of the Feature Extraction methods
3. Using the algorithms perform feature evaluation
4. Based on the Feature evaluation classify the URL as safe or malicious
5. Redirect the users to the provided URL if it is safe

3.4.2.3 Operating Environment

The operating environment for the system is as listed below.

1. Operating System: Windows 7 or above , Linux, macOS
2. Programming Language: Python
3. Programming Environment: Anaconda
4. Front End Environment: Flask

5. Dataset : Phishtank: Phishtank is a service Website dedicated for sharing phishing URLs.

URLhaus: URLhaus is a project aiming at sharing malicious URLs being used for malicious software distribution.

Alexa: Is a database ranking all websites according to their usefulness.

3.4.3 External Interface Requirements

3.4.3.1 User Interface

This Interface for the system is used to enter the URL on the output screen and the feature extraction for that particular website is done using the algorithms. The result of the detection is displayed in the white space below and along with that a dialog box is also displayed to redirect the user to the URL.

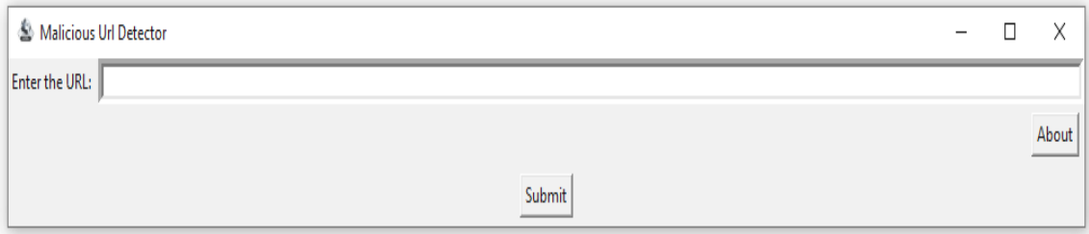


Figure. 3.4.3.1 User Interface of the URL Detector

3.4.3.2 Software Interfaces

1. Python is interpreted, high-level, general-purpose programming language. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. It uses the following libraries:

2. NumPy(Numerical Python) is a perfect tool for scientific computing and performing basic and advanced array operations.

3. Scipy library includes modules for linear algebra, integration, optimization, and statistics. Its main functionality was built upon NumPy, so its arrays make use of this library. SciPy works great for all kinds of scientific programming projects (science, mathematics, and engineering). It offers efficient numerical routines such as numerical optimization, integration, and others in sub-modules.

4. Pandas is a library created to help developers work with "labelled" and "relational" data intuitively. It's based on two main data structures: "Series" (one-dimensional, like a list of items) and "Data Frames" (two-dimensional, like a table with multiple columns). Pandas allows converting data structures to Data Frame objects, handling missing data, and adding/deleting columns from Data Frame, imputing missing files, and plotting data with histogram or plot box. It’s a must-have for data wrangling, manipulation, and visualization.

5. **GridSearchCV** lets you combine an estimator with a grid search preamble to tune hyper-parameters. The method picks the optimal parameter from the grid search and uses it with the estimator selected by the user. GridSearchCV inherits the methods from the classifier.

6. **Urlparse** This module defines a standard interface to break Uniform Resource Locator (URL) strings up in components (addressing scheme, network location, path etc.), to combine the components back into a URL string, and to convert a “relative URL” to an absolute URL given a “base URL.”

7. **SKlearn** It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

8. **Pygeoip** To map or co-relate IP addresses with physical location we will be using python's pyGeoIP module.

9. **Matplotlib** It is a comprehensive library for creating static, animated, and interactive visualizations in Python

10. **Tkinter** is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

11. **WebBrowser** module includes functions to open URLs in interactive browser applications. The module includes a registry of available browsers, in case multiple options are available on the system. It can also be controlled with the BROWSER environment variable.

3.5 COCOMO MODEL

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of lines of source code (KLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$\mathbf{E_i=a*(KLOC)b}$$

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

ORGANIC: Relatively small, simple software projects in which small teams with good application experience work to a set of less than rigid requirements.

SEMI-DETACHED: An intermediate, (in size and complexity), a software project in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements.

EMBEDDED: A software project that must be developed within a set of tight hardware, software and operation constraints.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

Basic COCOMO Model: The basic COCOMO model provides an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$Effort = a*KLOC^{b}$$

$$Duration = *effort^{d}$$

$$Staffing = effort/duration$$

THE ESTIMATION FOR OUR SYSTEM IS:

KLOC=2

	Organic	Semi-detached	Embedded
Variable A	2.4	3	3.6
Variable B	1.05	1.12	1.2
Variable C	2.5	2.5	2.5
Variable D	0.38	0.35	0.32
KLOC	2	2	2
Effort (In Person / Month)	4.969271634438612	6.520409175156349	8.27062815597865
Duration (In months)	4.597600684585728	4.818770091135695	4.915326003877275
Staffing (Recommended)	1.0808401980404643	1.3531272610724636	1.6826204710439692

4. SYSTEM DESIGN

4.1 System Architecture

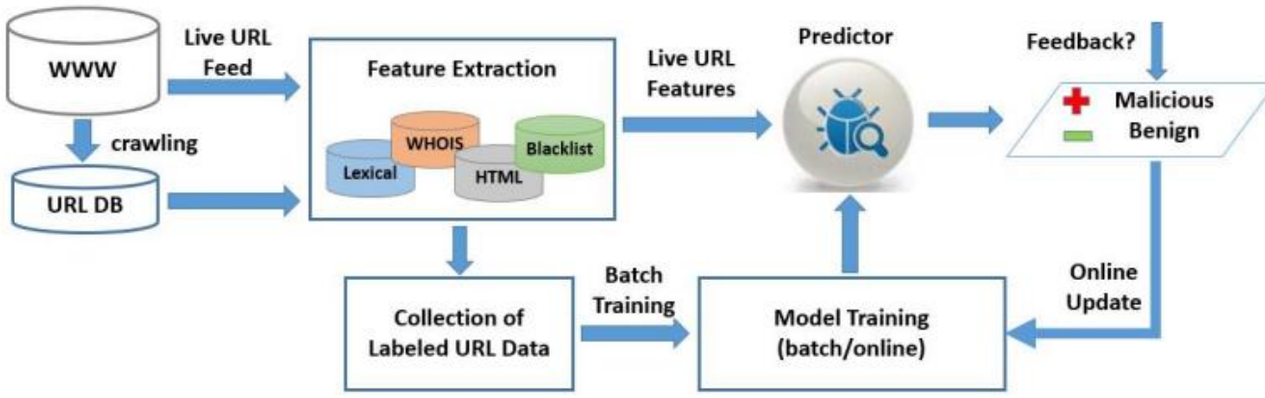


Figure.4.1.1 Architecture for URL Detector

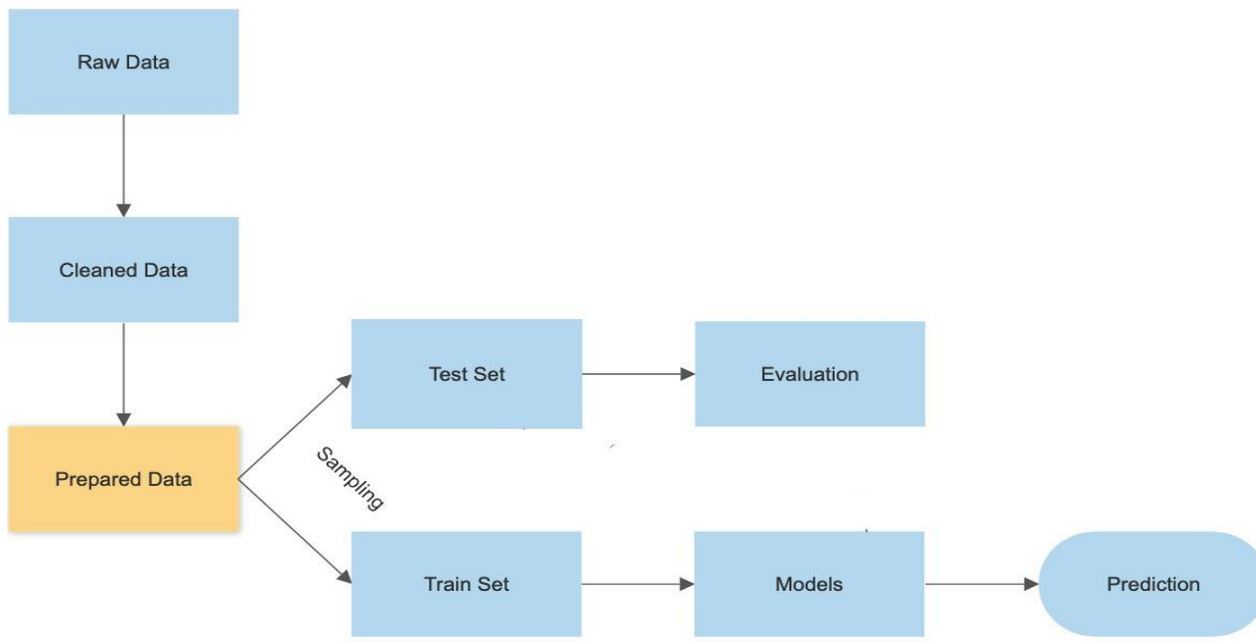


Figure.4.1.2 Flowchart for Data Set

4.2 Use Case Diagram for URL Detector

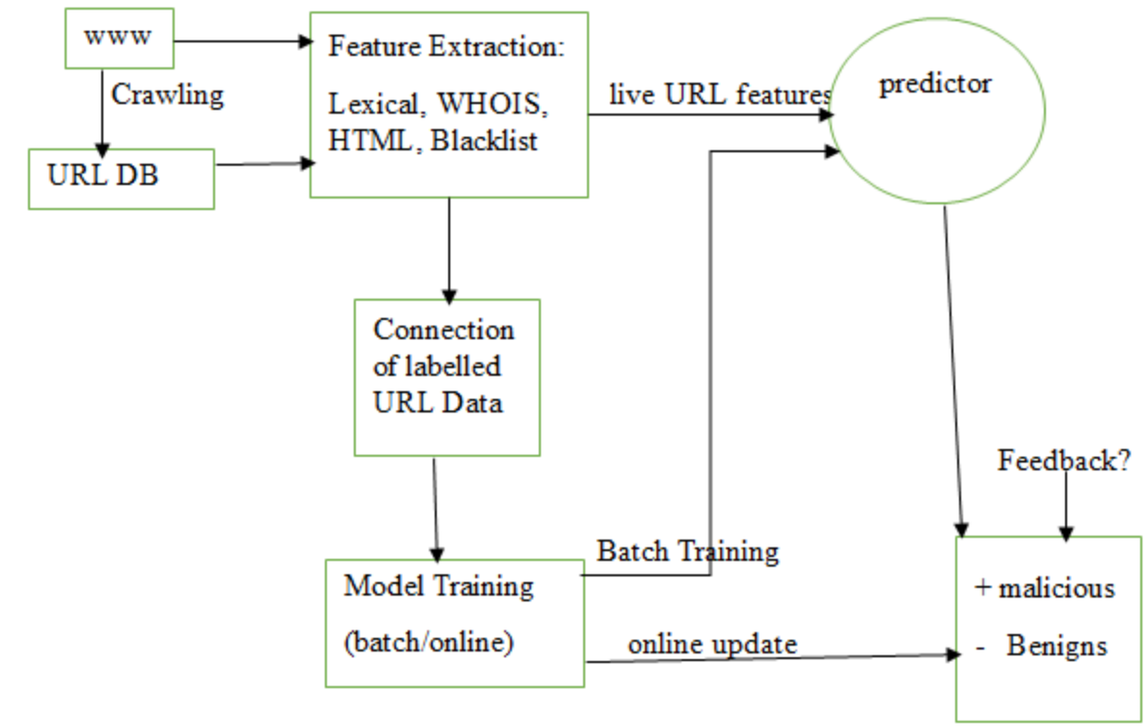


Figure.4.2. Use Case Diagram for URL detector

4.3 Activity Diagram of URL Detector

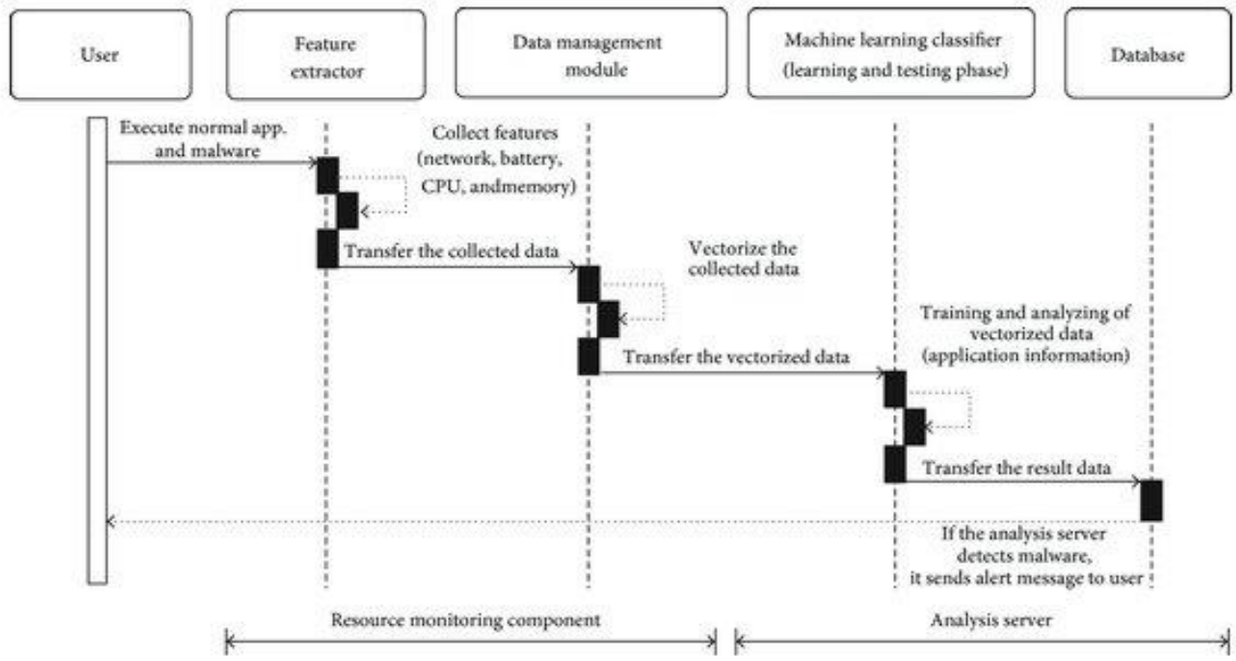


Figure.4.3. Activity Diagram to Malicious URL detector

5. IMPLEMENTATION

5.1 Data Set Used

PhisTank dataset containing data for 1,470 URLs with 15 features

	B	C	D	E	F	G	H	I	J	K	L	M	
1	rank_host	rank_country	ASNno	sec_sen_word_cnt	avg_token_length	No_of_dots	malicious	Length_of_url	avg_path_token	IPaddress_presence	Length_of_host	safebrowsing	URL
2	1	1	15169	0	4.333333333	1	0	17	0	0	10	0	http://google.com
3	2	2	32934	0	5	1	0	19	0	0	12	0	http://facebook.com
4	3	3	15169	0	4.666666667	1	0	18	0	0	11	0	http://youtube.com
5	4	4	36647	0	4	1	0	16	0	0	9	0	http://yahoo.com
6	6	1	23724	0	4	1	0	16	0	0	9	0	http://baidu.com
7	5	6	14907	0	5.333333333	1	0	20	0	0	13	0	http://wikipedia.org
8	7	2	17623	0	3	1	0	13	0	0	6	0	http://qq.com
9	10	8	20049	0	5	1	0	19	0	0	12	0	http://linkedin.com
10	11	15	8075	0	3.666666667	1	0	15	0	0	8	0	http://live.com
11	8	9	13414	0	4.666666667	1	0	18	0	0	11	0	http://twitter.com
12	12	5	16509	0	4.333333333	1	0	17	0	0	10	0	http://amazon.com
13	13	3	37963	0	4.333333333	1	0	17	0	0	10	0	http://taobao.com
14	9	13	15169	0	5	1	0	19	0	0	12	0	http://blogspot.com
15	17	1	15169	0	3.5	2	0	19	0	0	12	0	http://google.co.in
16	15	21	13768	0	5.333333333	1	0	20	0	0	13	0	http://wordpress.com
17	16	4	4808	0	3.25	2	0	18	0	0	11	0	http://sina.com.cn
18	29	1	13238	0	4	1	0	16	0	0	9	0	http://yandex.ru
19	26	1	24572	0	3.25	2	0	18	0	0	11	0	http://yahoo.co.jp
20	20	10	8075	0	3.666666667	1	0	15	0	0	8	0	http://bing.com
21	14	5	4808	0	4.333333333	1	0	17	0	0	10	0	http://hao123.com
22	27	1	15169	0	4	1	0	16	0	0	9	0	http://google.de
23	38	2	47541	0	3	1	0	13	0	0	6	0	http://vk.com
24	25	7	11643	0	3.666666667	1	0	15	0	0	8	0	http://ebay.com
25	18	6	45062	0	3.333333333	1	0	14	0	0	7	0	http://163.com

url_features

.....Includes more 1,351 rows

Table 5.1.1 Data Set used for detections

5.2 Front End Execution

5.2.1 GUI.py

```
from tkinter import *
import tkinter.messagebox as tkMessageBox
import trainer as tr
import webbrowser
import pandas
import main
import time
begin = time.time()


root = Tk()
```

```
root.title("Malicious Attack Detector")
img = PhotoImage(width=300,height=300)
data = ("{red red red red blue blue blue blue}")
root.attributes('-alpha',0.9)
root.iconbitmap(r'malware.ico')
#root.configure(background='thistle')
#root.geometry("800x100")
frame = Frame(root)
frame.pack()
bottomframe = Frame(root)
bottomframe.pack(side=BOTTOM)

L1 = Label(frame, text="Enter the URL: ")
L1.pack(side=LEFT)
E1 = Entry(frame, bd=5, width=150)
E1.pack(side=RIGHT)



        def submitCallBack():

            url = E1.get()

            main.process_test_url(url, 'test_features.csv')

            return_ans = tr.gui_caller('url_features.csv', 'test_features.csv')

            a = str(return_ans).split()

            print("-----")

            print("return_ans:",return_ans)

            print("-----")

            if int(a[1]) == 0:
```

```
        tkMessageBox.showinfo("URL Checker Result", "The URL " + url + " is Safe to Visit")

        new=1

        answer = tkMessageBox.askquestion("Redirect","Do you want to visit the url?")

        if answer == 'yes':

            #webbrowser.open(url=E1.get(), new=1)

            chrome_path = 'C:/Program Files (x86)/Google/Chrome/Application/chrome.exe %s'

            webbrowser.get(chrome_path).open(url=E1.get(),new=1)

    elif int(a[1]) == 1:

        tkMessageBox.showinfo("URL Checker Result", "The URL " + url + " is Malicious")

        answer_2 = tkMessageBox.askquestion("Redirect", "The url MALICIOUS, Do you still want to visit the url?")

        if answer_2=='yes':

            webbrowser.open(url=E1.get(),new=1)

    else:

        tkMessageBox.showinfo("URL Checker Result", "The URL " + url + " is Malware")

        tkMessageBox.showwarning("Warning","Cant Redirect, url contains a malware")



def about():
```

```
    tkMessageBox.showinfo("About","Authors:\nSuhaib Ahmed Sayeed \n"
                          +"Shajee Ahemd Musharaf \n"
                          + "Mohd Talha Haseeb")
B2 = Button(root, text="About", command=about)
B1 = Button(bottomframe, text="Submit", command=submitCallBack)
B2.pack(side=RIGHT, padx=5, pady=5)
B1.pack(side=RIGHT, padx=5,pady=5)
root.mainloop()
time.sleep(1)
end = time.time()
print(f"Total runtime of the program is {end - begin}")
```

Code Description: The following code gives us the UI for the user to enter the URL, display the results to classify URL as safe or malicious and based on the result redirects to the input URL.

5.3 Back end Execution

5.3.1 Main.py

```
import csv
import Feature_extraction as urlfeature
import trainer as tr
def resultwriter(feature, output_dest):
    flag = True
    with open(output_dest, 'w') as f:
```

```
        for item in feature:
            print("-------item------",item,"-----item-----")
            w = csv.DictWriter(f, item[1].keys())
            if flag:
                w.writeheader()
                flag = False
            w.writerow(item[1])
def process_URL_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.split(',')[0].strip()
            malicious_bool = line.split(',')[1].strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                ret_dict['malicious'] = malicious_bool
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)
```

```
def process_test_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)
# change
def process_test_url(url, output_dest):
    feature = []
    url = url.strip()
    if url != '':
        print
        'working on: ' + url  # showoff
        ret_dict = urlfeature.feature_extract(url)
        feature.append([url, ret_dict]);
```

```
    resultwriter(feature, output_dest)

def main():

    process_URL_list('URL.txt', 'url_features.csv')

    # process_test_list("query.txt",'query_features.csv')

    tr.train('url_features.csv', 'url_features.csv') # arguments:(input_training feature,test/query traning features)

    tr.train('url_features.csv', 'query_features.csv')

    # testing with urls in query.txt


print("main is running.....")
```

```
(base) D:\Malicious-URL-Detector-master\Malicious-URL-Detector-master>python gui.py
trainer is running ...
<urllib.request.OpenerDirector object at 0x0000026B077816D0>
feature extraction is running.......
main is running......
['https', 'www', 'youtube', 'com', ''] 5
```

Figure.5.3.1 Screenshot of running main class

Code Description: The following code helps to combines the GUI class with the backend i.e. the trainer and feature extraction class.

5.3.2 Trainer.py

```
import pandas
import pandas as pd
from sklearn import
preprocessing
```

```
from
sklearn.ensemble
import
RandomForestClassifi
er
from
sklearn.model_selecti
on import
train_test_split
import numpy
from sklearn import
svm
from
sklearn.model_selecti
on import
cross_validate as cv
#changed this package
below line
from sklearn import
model_selection
import
matplotlib.pylab as plt
import warnings
from sklearn.metrics
import
confusion_matrix

warnings.filterwarnin
gs("ignore",
category=Deprecation
Warning,
module="pandas",
lineno=570)
```

```
def
return_nonstring_col(
data_cols):
    cols_to_keep = []
    train_cols = []
    for col in data_cols:
        if col != 'URL'
and col != 'host' and
col != 'path':

cols_to_keep.append(
col)
            if col !=
'malicious' and col !=
'result':

train_cols.append(col)
    return
[cols_to_keep,
train_cols]



def
svm_classifier(train,
query, train_cols):
    clf = svm.SVC()

    train[train_cols] =
preprocessing.scale(tr
ain[train_cols])
    query[train_cols] =
preprocessing.scale(q
uery[train_cols])
```

```
print(clf.fit(train[train
_cols],
train['malicious']))
    scores =
cv.cross_val_score(clf
, train[train_cols],
train['malicious'],
cv=30)
    print('Estimated
score SVM: %0.5f
(+/- %0.5f)' %
(scores.mean(),
scores.std() / 2))

    query['result'] =
clf.predict(query[train
_cols])

    print(query[['URL',
'result']])


# Called from gui
def
forest_classifier_gui(t
rain, query,
train_cols):
    rf =
RandomForestClassifi
er(n_estimators=150)


print(rf.fit(train[train_
```

```
cols],
train['malicious']))

    query['result'] =
rf.predict(query[train_
cols])

    print(query[['URL',
'result']].head(2))
    return
query['result']


def
forest_classifier(train,
query, train_cols):
    rf =
RandomForestClassifi
er(n_estimators=150)


print(rf.fit(train[train_
cols],
train['malicious']))
    scores =
model_selection.cross
_val_score(rf,
train[train_cols],
train['malicious'],
cv=30)
    print('Estimated
score
RandomForestClassifi
er: %0.5f (+/- %0.5f)'
```

```
% (scores.mean(),
scores.std() / 2))

    query['result'] =
rf.predict(query[train_
cols])
    print(query[['URL',
'result']])
    return
query['result']


def train(db, test_db):
    query_csv =
pandas.read_csv(test_
db)
    cols_to_keep,
train_cols =
return_nonstring_col(
query_csv.columns)
    #
query=query_csv[cols
_to_keep]

    train_csv =
pandas.read_csv(db)
    cols_to_keep,
train_cols =
return_nonstring_col(t
rain_csv.columns)
    train =
train_csv[cols_to_kee
p]
```

```
svm_classifier(train_c
sv, query_csv,
train_cols)


forest_classifier(train
_csv, query_csv,
train_cols)


def gui_caller(db,
test_db):
    query_csv =
pandas.read_csv(test_
db)
    cols_to_keep,
train_cols =
return_nonstring_col(
query_csv.columns)
    #
query=query_csv[cols
_to_keep]

    train_csv =
pandas.read_csv(db)
    cols_to_keep,
train_cols =
return_nonstring_col(t
rain_csv.columns)
    train =
train_csv[cols_to_kee
p]
```

```
    return
forest_classifier(train
_csv, query_csv,
train_cols)
print("trainer is
running ...")350.3+ KB
```

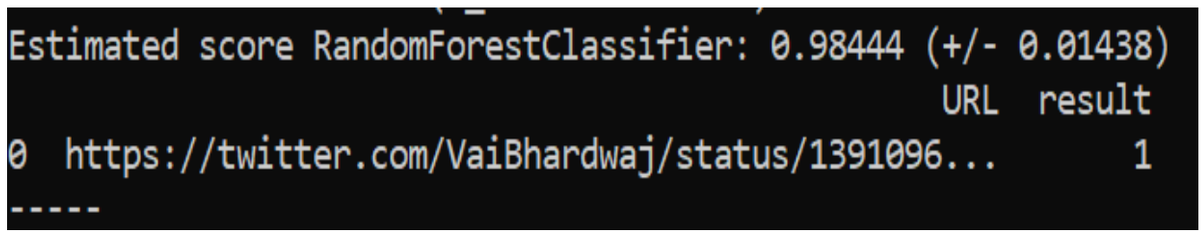


Figure. 5.3.2 Accuracy of the classifier model

5.3.3 **FeatureExtraction.py**

```
from urllib.parse import urlparse
import re
import urllib
import urllib.request  #changing import urllib2 to import urllib.request
from xml.dom import minidom
import csv
import pygeoip as pygeoip
import unicodedata
#import testingFeatureExtractionFunction as tfe

opener = urllib.request.build_opener()  #changing urllib2.build_opener() to
urllib.request.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
print(opener)

nf = -1


def Tokenise(url):
    if url == '':
        return [0, 0, 0]
    token_word = re.split('\W+', url)
    print("*****token_word:",token_word)
    no_ele = sum_len = largest = 0
    for ele in token_word:
```

```
        l = len(ele)
        sum_len += l
        if l > 0:  ## for empty element exclusion in average length
            no_ele += 1
        if largest < l:
            largest = l
    try:

        return [float(sum_len) / no_ele, no_ele, largest]
    except:
        return [0, no_ele, largest]


def find_ele_with_attribute(dom, ele, attribute):
    print("*******************************")#remove
    #see here see here next you should change here
    for subelement in dom.getElementsByTagName(ele):
        print(subelement)
        if subelement.hasAttribute(attribute):
            print('====',subelement.attributes[attribute].value,'====')
            return subelement.attributes[attribute].value
    return nf


#this function is modified as below
def sitepopularity(host):
    xmlpath = 'http://data.alexa.com/data?cli=10&dat=snbamz&url=' + host
    print(xmlpath," this is the original url:
http://data.alexa.com/data?cli=10&dat=snbamz&url= that is replaced by new url
above")
    if(xmlpath!=" "):
        xml = urllib.request.urlopen(xmlpath)
        dom = minidom.parse(xml)
        rank_host = find_ele_with_attribute(dom, 'REACH', 'RANK')
        print("********rank host:**********",rank_host)
        rank_country = find_ele_with_attribute(dom, 'COUNTRY', 'RANK')
        print("********rank country:**********",rank_country)
        return [rank_host, rank_country]

    else:
        return [nf, nf]
'''


def sitepopularity(host):
    rank_host= tfe.feaEx(host,'REACH','RANK')
    print("********rank host:**********",rank_host)
```

```
    rank_country= tfe.feaEx(host,'COUNTRY','RANK')
    print("********rank country:**********",rank_country)
    return [rank_host, rank_country]
'''
def Security_sensitive(tokens_words):
    sec_sen_words = ['confirm', 'account', 'banking', 'secure', 'ebayisapi', 'webscr', 'login',
'signin']
    cnt = 0
    for ele in sec_sen_words:
        if (ele in tokens_words):
            cnt += 1;

    return cnt


def exe_in_url(url):
    if url.find('.exe') != -1:
        return 1
    return 0


def Check_IPaddress(tokens_words):
    cnt = 0;
    for ele in tokens_words:
        if str(ele).isnumeric():
            cnt += 1
        else:
            if cnt >= 4:
                return 1
            else:
                cnt = 0;
    if cnt >= 4:
        return 1
    return 0


def getASN(host):
    try:
        g = pygeoip.GeoIP('GeoIPASNum.dat')
        asn = int(g.org_by_name(host).split()[0][2:])
        return asn
    except:
        return nf


'''def web_content_features(url):
```

```
    wfeatures={}
    total_cnt=0
    try:
        source_code = str(opener.open(url))
        #print source_code[:500]

        wfeatures['src_html_cnt']=source_code.count('<html')
        wfeatures['src_hlink_cnt']=source_code.count('<a href=')
        wfeatures['src_iframe_cnt']=source_code.count('<iframe')
        #suspicioussrc_ javascript functions count

        wfeatures['src_eval_cnt']=source_code.count('eval(')
        wfeatures['src_escape_cnt']=source_code.count('escape(')
        wfeatures['src_link_cnt']=source_code.count('link(')
        wfeatures['src_underescape_cnt']=source_code.count('underescape(')
        wfeatures['src_exec_cnt']=source_code.count('exec(')
        wfeatures['src_search_cnt']=source_code.count('search(')

        for key in wfeatures:
            if(key!='src_html_cnt' and key!='src_hlink_cnt' and key!='src_iframe_cnt'):
                total_cnt+=wfeatures[key]
        wfeatures['src_total_jfun_cnt']=total_cnt

    except Exception, e:
        print "Error"+str(e)+" in downloading page "+url
        default_val=nf

        wfeatures['src_html_cnt']=default_val
        wfeatures['src_hlink_cnt']=default_val
        wfeatures['src_iframe_cnt']=default_val
        wfeatures['src_eval_cnt']=default_val
        wfeatures['src_escape_cnt']=default_val
        wfeatures['src_link_cnt']=default_val
        wfeatures['src_underescape_cnt']=default_val
        wfeatures['src_exec_cnt']=default_val
        wfeatures['src_search_cnt']=default_val
        wfeatures['src_total_jfun_cnt']=default_val

    return wfeatures"'


def safebrowsing(url):
    api_key = "ABQIAAAA8C6Tfr7tocAe04vXo5uYqRTEYoRzLFR0-
nQ3fRl5qJUqcubbrw"
    name = "URL_check"
    ver = "1.0"
```

```
    req = {}
    req["client"] = name
    req["apikey"] = api_key
    req["appver"] = ver
    req["pver"] = "3.0"
    req["url"] = url  # change to check type of url

    try:
        params = urllib.urlencode(req)
        req_url = "https://sb-ssl.google.com/safebrowsing/api/lookup?" + params
        res = urllib.request.urlopen(req_url)
        # print res.code
        # print res.read()
        if res.code == 204:
            print
            "safe"
            return 0
        elif res.code == 200:
            print
            "The queried URL is either phishing, malware or both, see the response body for
the specific type."
            return 1
        elif res.code == 204:
            print
            "The requested URL is legitimate, no response body returned."
        elif res.code == 400:
            print
            "Bad Request The HTTP request was not correctly formed."
        elif res.code == 401:
            print
            "Not Authorized The apikey is not authorized"
        else:
            print
            "Service Unavailable The server cannot handle the request. Besides the normal
server failures, it could also indicate that the client has been throttled by sending too
many requests"
    except:
        return -1

def having_ip_address(url):
    ip_address_pattern = ipv4_pattern + "|" + ipv6_pattern
    match = re.search(ip_address_pattern, url)
    return -1 if match else 1
```

```
def url_length(url):
    if len(url) < 54:
        return 1
    if 54 <= len(url) <= 75:
        return 0
    return -1


def shortening_service(url):
    match = re.search(shortening_services, url)
    return -1 if match else 1


def having_at_symbol(url):
    match = re.search('@', url)
    return -1 if match else 1


def double_slash_redirecting(url):
    # since the position starts from 0, we have given 6 and not 7 which is according to the
document.
    # It is convenient and easier to just use string search here to search the last occurrence
instead of re.
    last_double_slash = url.rfind('//')
    return -1 if last_double_slash > 6 else 1


def prefix_suffix(domain):
    match = re.search('-', domain)
    return -1 if match else 1


def having_sub_domain(url):
    # Here, instead of greater than 1 we will take greater than 3 since the greater than 1
condition is when www and
    # country domain dots are skipped
    # Accordingly other dots will increase by 1
    if having_ip_address(url) == -1:
        match = re.search(
            '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-
5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
            '([01]?\\d\\d?|2[0-4]\\d|25[0-5]))|(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}',
            url)
        pos = match.end()
        url = url[pos:]
    num_dots = [x.start() for x in re.finditer(r'\.', url)]
    if len(num_dots) <= 3:
```

```
        return 1
    elif len(num_dots) == 4:
        return 0
    else:
        return -1


def domain_registration_length(domain):
    expiration_date = domain.expiration_date
    today = time.strftime('%Y-%m-%d')
    today = datetime.strptime(today, '%Y-%m-%d')

    registration_length = 0
    # Some domains do not have expiration dates. This if condition makes sure that the
expiration date is used only
    # when it is present.
    if expiration_date:
        registration_length = abs((expiration_date - today).days)
    return -1 if registration_length / 365 <= 1 else 1


def favicon(wiki, soup, domain):
    for head in soup.find_all('head'):
        for head.link in soup.find_all('link', href=True):
            dots = [x.start() for x in re.finditer(r'\.', head.link['href'])]
            return 1 if wiki in head.link['href'] or len(dots) == 1 or domain in
head.link['href'] else -1
    return 1


def https_token(url):
    match = re.search(http_https, url)
    if match and match.start() == 0:
        url = url[match.end():]
    match = re.search('http|https', url)
    return -1 if match else 1


def request_url(wiki, soup, domain):
    i = 0
    success = 0
    for img in soup.find_all('img', src=True):
        dots = [x.start() for x in re.finditer(r'\.', img['src'])]
        if wiki in img['src'] or domain in img['src'] or len(dots) == 1:
            success = success + 1
        i = i + 1
```

```
    for audio in soup.find_all('audio', src=True):
        dots = [x.start() for x in re.finditer(r'\.', audio['src'])]
        if wiki in audio['src'] or domain in audio['src'] or len(dots) == 1:
            success = success + 1
        i = i + 1

    for embed in soup.find_all('embed', src=True):
        dots = [x.start() for x in re.finditer(r'\.', embed['src'])]
        if wiki in embed['src'] or domain in embed['src'] or len(dots) == 1:
            success = success + 1
        i = i + 1

    for i_frame in soup.find_all('i_frame', src=True):
        dots = [x.start() for x in re.finditer(r'\.', i_frame['src'])]
        if wiki in i_frame['src'] or domain in i_frame['src'] or len(dots) == 1:
            success = success + 1
        i = i + 1

    try:
        percentage = success / float(i) * 100
    except:
        return 1

    if percentage < 22.0:
        return 1
    elif 22.0 <= percentage < 61.0:
        return 0
    else:
        return -1


def url_of_anchor(wiki, soup, domain):
    i = 0
    unsafe = 0
    for a in soup.find_all('a', href=True):
        # 2nd condition was 'JavaScript ::void(0)' but we put JavaScript because the space
between javascript and ::
        # might not be
        # there in the actual a['href']
        if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto" in a['href'].lower() or
not (
                wiki in a['href'] or domain in a['href']):
            unsafe = unsafe + 1
        i = i + 1
        # print a['href']
```

```
    try:
        percentage = unsafe / float(i) * 100
    except:
        return 1
    if percentage < 31.0:
        return 1
        # return percentage
    elif 31.0 <= percentage < 67.0:
        return 0
    else:
        return -1


# Links in <Script> and <Link> tags
def links_in_tags(wiki, soup, domain):
    i = 0
    success = 0
    for link in soup.find_all('link', href=True):
        dots = [x.start() for x in re.finditer(r'\.', link['href'])]
        if wiki in link['href'] or domain in link['href'] or len(dots) == 1:
            success = success + 1
        i = i + 1

    for script in soup.find_all('script', src=True):
        dots = [x.start() for x in re.finditer(r'\.', script['src'])]
        if wiki in script['src'] or domain in script['src'] or len(dots) == 1:
            success = success + 1
        i = i + 1
    try:
        percentage = success / float(i) * 100
    except:
        return 1

    if percentage < 17.0:
        return 1
    elif 17.0 <= percentage < 81.0:
        return 0
    else:
        return -1


# Server Form Handler (SFH)
# Have written conditions directly from word file..as there are no sites to test ######
def sfh(wiki, soup, domain):
    for form in soup.find_all('form', action=True):
        if form['action'] == "" or form['action'] == "about:blank":
```

```
            return -1
        elif wiki not in form['action'] and domain not in form['action']:
            return 0
        else:
            return 1
    return 1


# Mail Function
# PHP mail() function is difficult to retrieve, hence the following function is based on
mailto
def submitting_to_email(soup):
    for form in soup.find_all('form', action=True):
        return -1 if "mailto:" in form['action'] else 1
    # In case there is no form in the soup, then it is safe to return 1.
    return 1


def abnormal_url(domain, url):
    hostname = domain.name
    match = re.search(hostname, url)
    return 1 if match else -1


# IFrame Redirection
def i_frame(soup):
    for i_frame in soup.find_all('i_frame', width=True, height=True, frameBorder=True):
        # Even if one iFrame satisfies the below conditions, it is safe to return -1 for this
method.
        if i_frame['width'] == "0" and i_frame['height'] == "0" and i_frame['frameBorder']
== "0":
            return -1
        if i_frame['width'] == "0" or i_frame['height'] == "0" or i_frame['frameBorder'] ==
"0":
            return 0
    # If none of the iframes have a width or height of zero or a frameBorder of size 0,
then it is safe to return 1.
    return 1


def age_of_domain(domain):
    creation_date = domain.creation_date
    expiration_date = domain.expiration_date
    ageofdomain = 0
    if expiration_date:
        ageofdomain = abs((expiration_date - creation_date).days)
    return -1 if ageofdomain / 30 < 6 else 1
```

```
def web_traffic(url):
    try:
        rank = \

bs4.BeautifulSoup(urllib.urlopen("http://data.alexa.com/data?cli=10&dat=s&url=" +
url).read(), "xml").find(
                "REACH")['RANK']
    except TypeError:
        return -1
    rank = int(rank)
    return 1 if rank < 100000 else 0


def google_index(url):
    site = search(url, 5)
    return 1 if site else -1


def statistical_report(url, hostname):
    try:
        ip_address = socket.gethostbyname(hostname)
    except:
        return -1
    url_match = re.search(

r'at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|sweddy\.com|myjino\.ru|9
6\.lt|ow\.ly', url)
    ip_match = re.search(

'146\.112\.61\.108|213\.174\.157\.151|121\.50\.168\.88|192\.185\.217\.116|78\.46\.211\.1
58|181\.174\.165\.13|46\.242\.145\.103|121\.50\.168\.40|83\.125\.22\.219|46\.242\.145\.9
8|'

'107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.151\.148\.1
08|107\.151\.148\.109|119\.28\.52\.61|54\.83\.43\.69|52\.69\.166\.231|216\.58\.192\.225|'

'118\.184\.25\.86|67\.208\.74\.71|23\.253\.126\.58|104\.239\.157\.210|175\.126\.123\.219
|141\.8\.224\.221|10\.10\.10\.10|43\.229\.108\.32|103\.232\.215\.140|69\.172\.201\.153|'

'216\.218\.185\.162|54\.225\.104\.146|103\.243\.24\.98|199\.59\.243\.120|31\.170\.160\.6
1|213\.19\.128\.77|62\.113\.226\.131|208\.100\.26\.234|195\.16\.127\.102|195\.16\.127\.1
57|'
```

```
'34\.196\.13\.28|103\.224\.212\.222|172\.217\.4\.225|54\.72\.9\.51|192\.64\.147\.141|198\
.200\.56\.183|23\.253\.164\.103|52\.48\.191\.26|52\.214\.197\.72|87\.98\.255\.18|209\.99\
.17\.27|'

'216\.38\.62\.18|104\.130\.124\.96|47\.89\.58\.141|78\.46\.211\.158|54\.86\.225\.156|54\.
82\.156\.19|37\.157\.192\.102|204\.11\.56\.48|110\.34\.231\.42',
        ip_address)
    if url_match:
        return -1
    elif ip_match:
        return -1
    else:
        return 1


def get_hostname_from_url(url):
    hostname = url
    # TODO: Put this pattern in patterns.py as something like - get_hostname_pattern.
    pattern = "https://|http://|www.|https://www.|http://www."
    pre_pattern_match = re.search(pattern, hostname)

    if pre_pattern_match:
        hostname = hostname[pre_pattern_match.end():]
        post_pattern_match = re.search("/", hostname)
        if post_pattern_match:
            hostname = hostname[:post_pattern_match.start()]

    return hostname


def feature_extract(url_input):
    Feature = {}
    tokens_words = re.split('\W+', url_input)  # Extract bag of words stings delimited by
(.,/,?,,=,-,_)
    print(tokens_words,len(tokens_words))

    # token_delimit1=re.split('[./?=-_]',url_input)
    # print token_delimit1,len(token_delimit1)

    obj = urlparse(url_input)
    host = obj.netloc #<scheme>://<netloc>/<path>;<params>?<query>#<fragment>
    #netloc is what the first level domain (FLD)
    path = obj.path
    print(host)
    print(path)

    Feature['URL'] = url_input
```

```
    print("host host:",url_input)

    Feature['rank_host'], Feature['rank_country'] = sitepopularity(url_input)

    Feature['host'] = obj.netloc
    Feature['path'] = obj.path

    Feature['Length_of_url'] = len(url_input)
    Feature['Length_of_host'] = len(host)
    Feature['No_of_dots'] = url_input.count('.')

    Feature['avg_token_length'], Feature['token_count'], Feature['largest_token'] =
Tokenise(url_input)
    Feature['avg_domain_token_length'], Feature['domain_token_count'],
Feature['largest_domain'] = Tokenise(host)
    Feature['avg_path_token'], Feature['path_token_count'], Feature['largest_path'] =
Tokenise(path)

    Feature['sec_sen_word_cnt'] = Security_sensitive(tokens_words)
    Feature['IPaddress_presence'] = Check_IPaddress(tokens_words)

    print
    host
    print
    getASN(host)
    Feature['exe_in_url'] = exe_in_url(url_input)
    Feature['ASNno'] = getASN(host)
    Feature['safebrowsing'] = safebrowsing(url_input)
    """wfeatures=web_content_features(url_input)

    for key in wfeatures:
        Feature[key]=wfeatures[key]
    """
    # debug
    # for key in Feature:
    #     print key +':'+str(Feature[key])
    print(Feature)
    return Feature


print("feature extraction is running.....")
```

```
{
    "URL": "https://twitter.com/VaiBhardwaj/status/1391096909209743374?ref_src=twsrc%5Egoogle%7Ctwcamp%
5Eserp%7Ctwgr%5Etweet",
    "rank_host": "49",
    "rank_country": "32",
    "host": "twitter.com",
    "path": "/VaiBhardwaj/status/1391096909209743374",
    "Length_of_url": 112,
    "Length_of_host": 11,
    "No_of_dots": 1,
    "avg_token_length": 7.538461538461538,
    "token_count": 13,
    "largest_token": 19,
    "avg_domain_token_length": 5.0,
    "domain_token_count": 2,
    "largest_domain": 7,
    "avg_path_token": 12.0,
    "path_token_count": 3,
    "largest_path": 19,
    "sec_sen_word_cnt": 0,
    "IPaddress_presence": 0,
    "exe_in_url": 0,
    "ASNno": -1,
    "safebrowsing": -1
}
```


Table 5.3.3 Features Extracted in JSON format

Code Description: The following code consists a set of methods which are defined to extract the particular feature of the URL.

6. TESTING

Test case id	Test cases	Steps to be executed	Expected result	Actual result	Pass/ Fail
1	URL field accepts input	Entering any URL	URL field is expected to accept any text	User was able to populate the URL field with any text in the form of URL	Pass

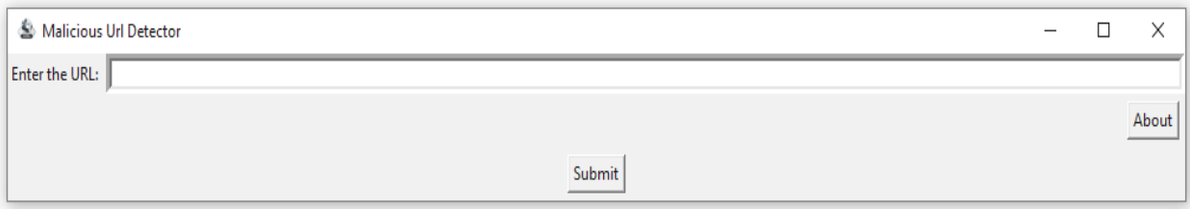


Figure.6.1. Entering URL

Test case id	Test cases	Steps to be executed	Expected result	Actual result	Pass/ Fail
2	Input field accepts URL of any length	Entering of URL of any length	Input field is expected to accept URL of any length	URL of any length can be populated in the input field	Pass

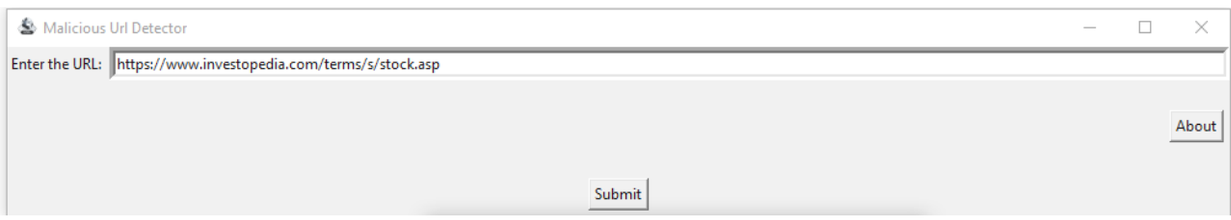


Figure.6.2. Entering URL of any length

Test case id	Test cases	Steps to be executed	Expected result	Actual result	Pass/ Fail
3	On click button – “ABOUT”	Click the about button	Expected to display the authors of project. on click	Authors were displayed on click of about button	Pass

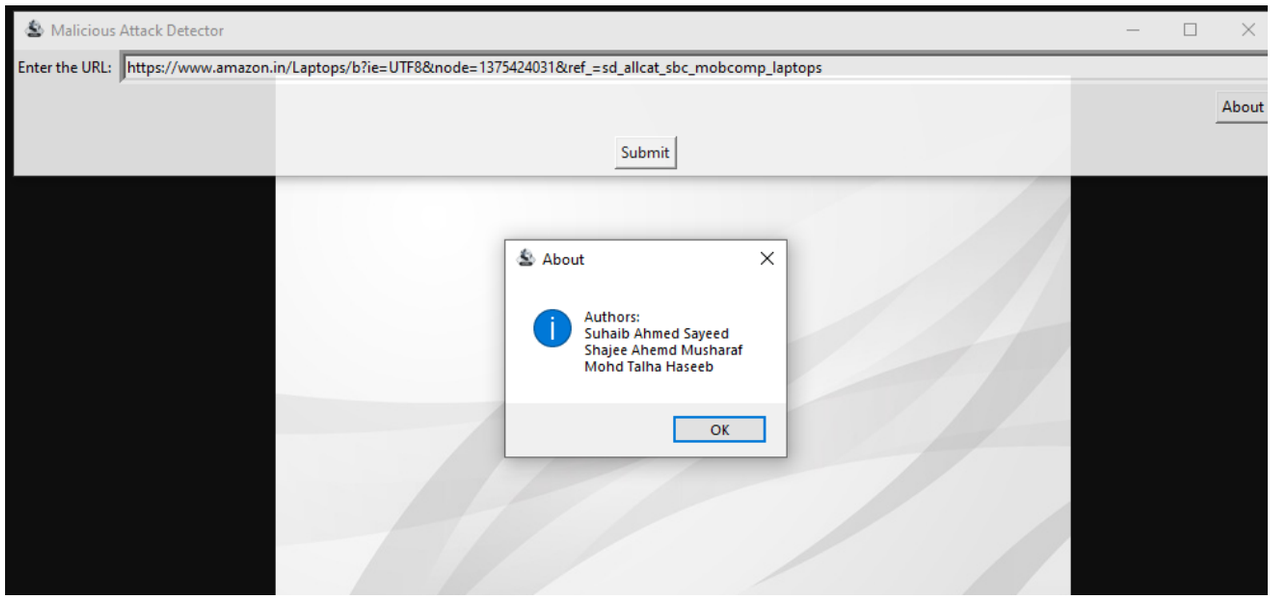


Figure.6.3. Implementation of ABOUT button

Test case id	Test cases	Steps to be executed	Expected result	Actual result	Pass/ Fail
4	On click button – “SUBMIT”	Click the submit button	Appearance of result dialog box is expected	Result dialog box appears	Pass

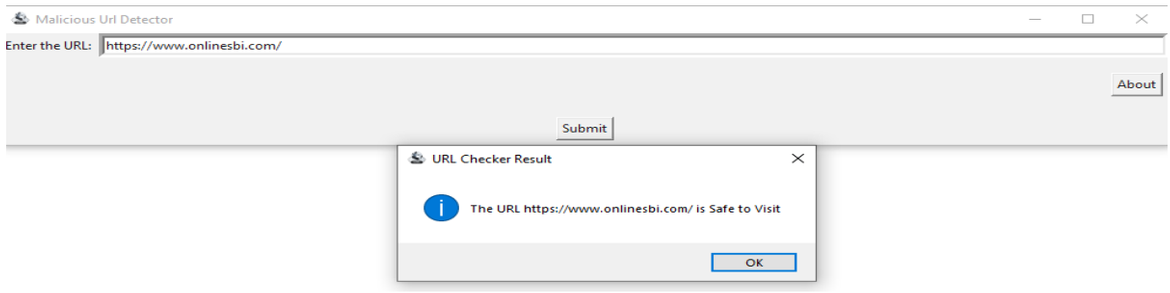


Figure.6.4. Implementation of SUBMIT button

Test case id	Test cases	Steps to be executed	Expected result	Actual result	Pass/ Fail
5	Result should display – “URL xyz is safe to visit/ the URL is malware or malicious”	Click the submit button	Depending on the URL, specific message is expected	Result box displays the message correctly	Pass

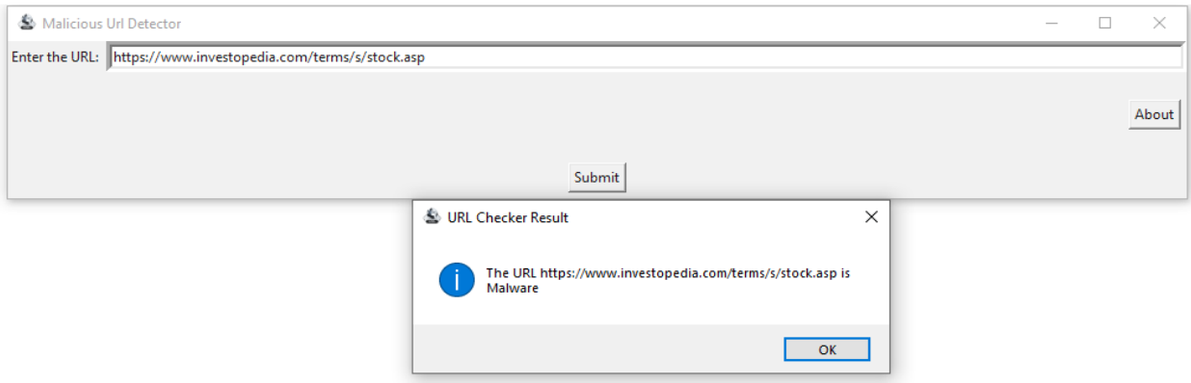


Figure.6.5. Result to be displayed

Test case id	Test cases	Steps to be executed	Expected result	Actual result	Pass/ Fail
6	On click button – “OK”	Click the OK button	Depending on the URL, user is asked if he wants to redirect to the given input.	Respective page is loaded in chrome successfully.	Pass

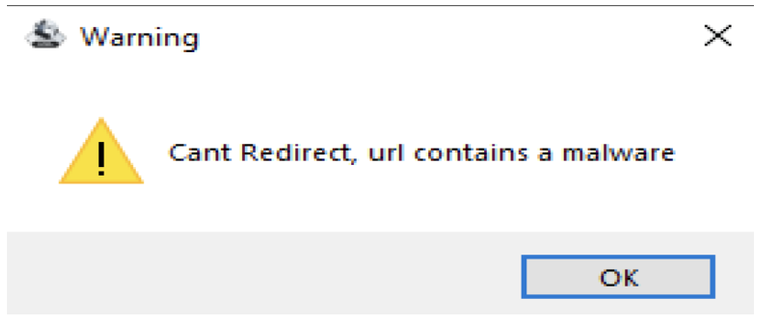


Figure.6.6. Output on malware URL detection

7. OUTPUT SCREENS

7.1 **Home Page** of the Employee Attrition Predictor

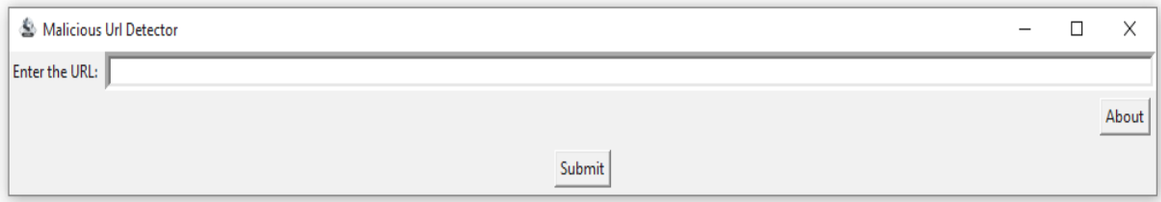


Figure.7.1. Main UI for user input

The above output screen indicates the main user interface of Malicious Attack Detector for the user to enter the URL

7.2 URL which is Safe to visit

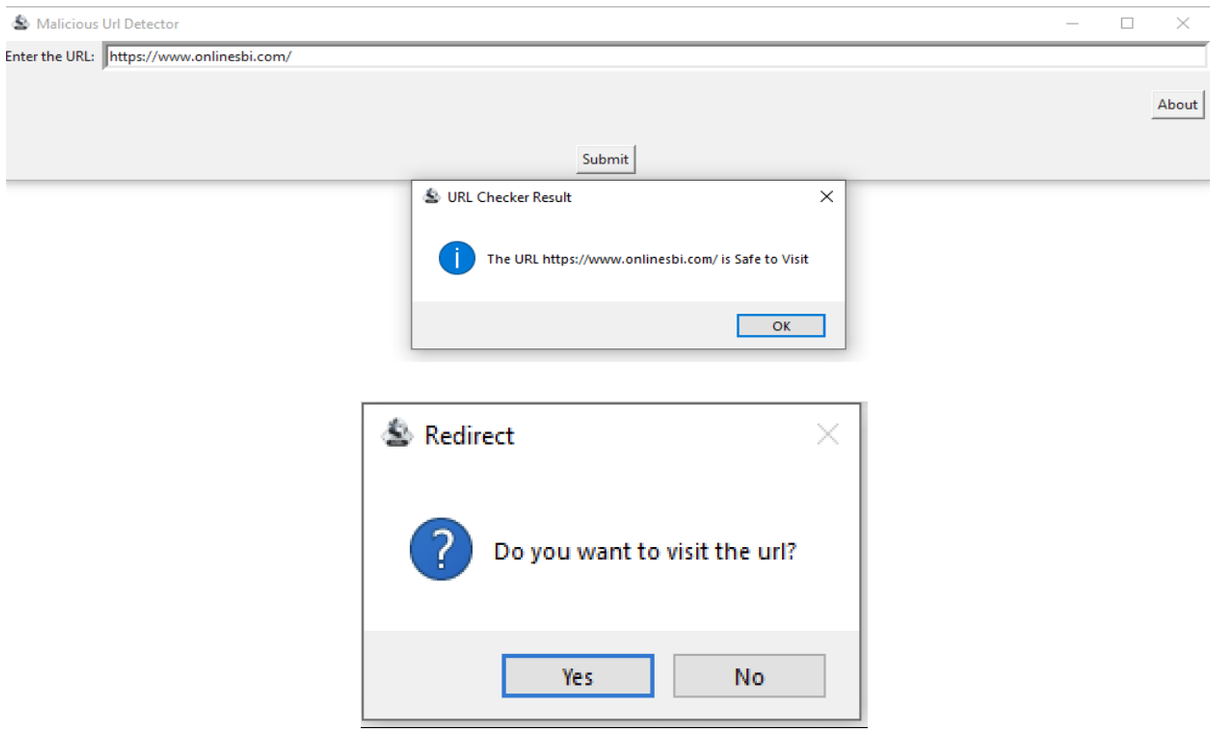


Figure.7.2. Output displayed when the URL is safe

The above output is displayed if the website is classified as SAFE, further it asks the user if he wants to visit the provided URL.

7.3 Website that is malicious

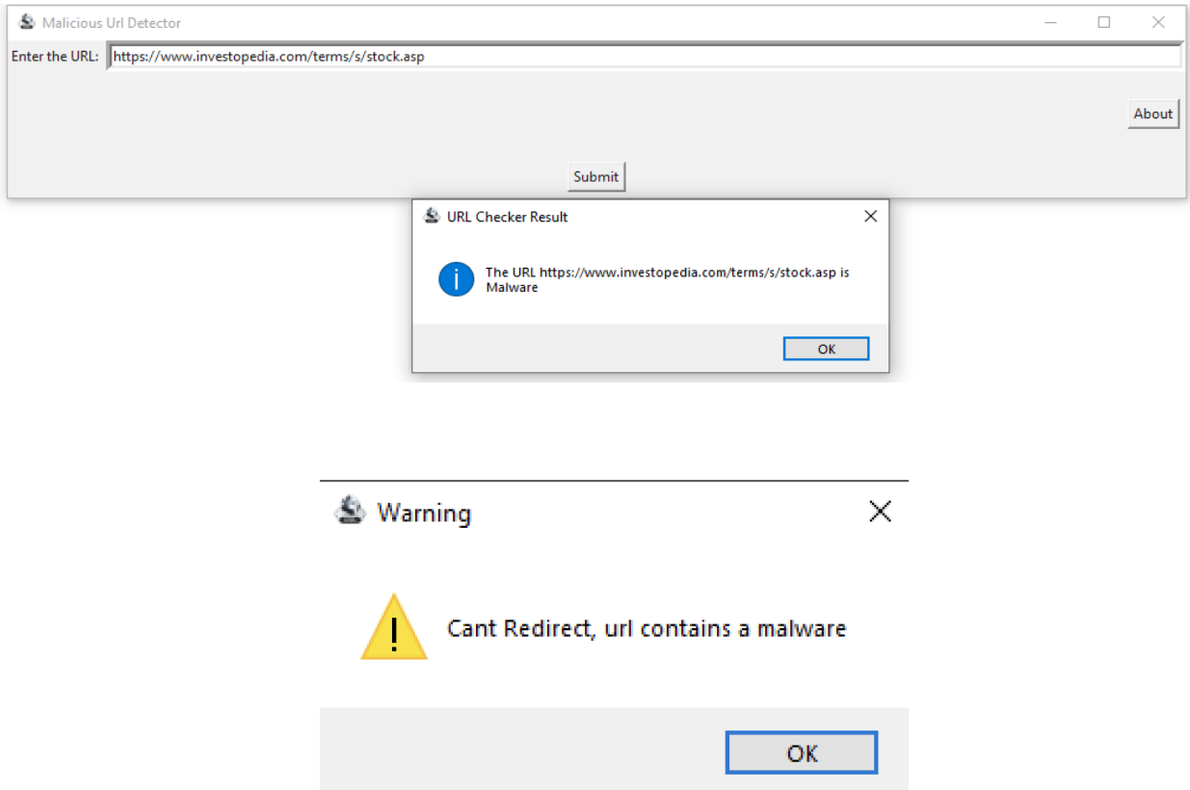


Figure.7.3. Output displayed if the URL contains malware

The above output is displayed if the website is classified as MALWARE, further warns the user the provided URL cant be redirected.

CONCLUSION

Despite existing defenses, malicious Web sites remain a scourge of the Internet. To protect end users from visiting these sites, we can attempt to identify suspicious URLs by analyzing their lexical and host-based features. A particular challenge in this domain is that URL classifiers must operate in a dynamic landscape; one in which criminals are constantly evolving new strategies to counter our defenses. To prevail in this contest, we need algorithms that continually adapt to new examples and features. In this project, we experimented with different approaches for detecting malicious URLs with an eye toward ultimately deploying a real-time system. In this work, we have described how a machine can able to judge the URLs based upon the given feature set. Specifically, we described the feature sets and an approach for classifying the given the feature set for malicious URL detection. When traditional method fall short in detecting the new malicious URLs on its own, our proposed method can be augmented with it and is expected to provide improved results. Here in this work, we proposed the feature set which can able to classify the URLs.

Future work is to fine tuning the machine learning algorithm that will produce the better result by utilizing the given feature set. Adding to that the open question is how we can handle the huge number of URLs whose features set will evolve over time. Certain efforts have to be made in that direction so as to come up with the more robust feature set which can change with respect to the evolving changes.

Future Enhancement

Some limitations attached to the present study could motivate further research; e.g., our team did not exhaustively explore all the network configurations and hyper parameters available for DNNs which may potentially improve their performances. While these improvements may lead to succeeding RF's reported accuracy, there is an impact on training and testing times as well as additional drawback of over-fitting models thus reducing their real-word generalizability.

Finally, we did not deploy and investigate the efficacy of the models with further experiments as explored in we leave this as our future research work. Importantly, we feel that more research on this front is needed to better bridge the gap between academic research and industrial applications with the goal of reducing detrimental economic impacts malicious URLs and other various formats such as files, documents on organizations in various industries.

Automatic detection of malicious URLs through browsers extensions can also be implemented in our future work. Additionally, interfacing our tool through a web application and graphically representing all the malwares present with some incident response techniques will be our top notch priority.

References

[1] Doyen Sahoo, Chenghao Liu and Steven C.H. Hoi. 2019; Malicious URL Detection using Machine Learning: A Survey. 1, 1 (August 2019)

[2] Alhanoof Faiz Alwaghid and Nurul I. Sarkar 2; Exploring Malware Behavior of Webpages Using Machine Learning Technique: An Empirical Study

[3] Malicious URL Detection : A Survey- Eint Sandi Aung, Hayato YAMANA, Department of Computer Science and Communication Engineering, Graduate School of Fundamental Science and Engineering, Waseda University, Tokyo, 169-8555, Japan.

[4] Classification of Malware Attacks Using Machine Learning In Decision
Treehttps://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

[5] Detecting Malicious Web Links and Identifying Their Attack Types – Hyunsang Choi Korea University Seoul, Korea realchs@korea.ac.kr Bin B. Zhu Microsoft Research Asia Beijing, China binzhu@microsoft.com Heejo Lee Korea University Seoul, Korea heejo@korea.ac.kr

[6] Malicious web page detection based on on-linelearning algorithm - WEN ZHANG, YU-XIN DING, YAN TANG, BIN ZHAO. Department of Computer Sciences and Technology, Harbin Institute of Technology Shenzhen Graduate School Shenzhen, China

[7] Detecting Malicious URLs using Machine Learning Techniques- Frank Vanhoenshoven∗, Gonzalo Napoles ´ ∗, Rafael Falcon†, Koen Vanhoof∗ and Mario Koppen ¨ ∗Universiteit Hasselt Campus Diepenbeek Agoralaan Gebouw D, BE3590 Diepenbeek, Belgium

[8] Adaptive Malicious URL Detection: Learning in the Presence of Concept Drifts 1st Guolin Tan Institute of Information Engineering, CAS University of Chinese Academy of Sciences Beijing, China tanguolin@iie.ac.cn

[9] Machine Learning & Concept Drift based Approach for Malicious Website Detection Siddharth Singhal, Utkarsh Chawla Dept. of Computer Science & Engineering Delhi Technological University New Delhi, India

APPENDIX 1

RELEVANCE OF PROJECT TO POs / PSOs

Title of Project	MALWARE ATTACK DETECTION
Implementation Details	PYTHON
Cost (hardware or software cost)	-NA-
Type (Application, Product, Research, Review, etc.)	APPLICATION

Mapping with POs and PSOs with Justification													
Relevance	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PSO 1	PSO 2
	1	2	3	-	2	-	-	1	2	3	-	1	1
Program Outcomes Justification	PO1: Engineering Knowledge: SDLC phases are followed in the execution of the project. PO2: Problem Analysis: The different steps involved in Problem Analysis for formulation of the solution i.e. literature survey and use of fundamental subject knowledge has been followed. PO3: Design/Development of solutions – Existing strategy has been enhanced using the design principles. PO5: Modern Tool Usage: Jupyter and Spyder Notebook are used. PO8: Ethics: Students have followed professional ethics during the various stages of Project completion. PO9: Individual and Team Work: Students have worked both in individual as well as team capacity during the various stages of project work. PO10: Communication: Effective communication with team members and during project reviews, project seminar and viva-voce has been exhibited. PO12: Lifelong Learning. The project carried out gives the students scope to continue the work in the Computer Vision area in future.												
Program Specific Outcomes Justification	PSO1: Use of open source software Jupyter and Spyder Notebook (Anaconda Navigator). PSO2: Web browser is used to display the predictor tool to perform predictions on employee attrition.												

APPENDIX 2 GANTT CHART

MALWARE ATTACK DETECTION

PLAN OF WORK

Activity	Plan start (No. of weeks)	Plan duration (No. of weeks)	Actual start (No. of weeks)	Actual Duration (No. of weeks)	Percent complete
Problem Definition	1	2	1	3	100%
Abstract	3	1	3	1	100%
Literature Survey	3	2	3	3	100%
System Analysis	5	3	5	4	100%
REVIEW I	8	1	8	1	100%
System Design	9	2	9	3	100%
Implementation	11	6	11	8	100%
Testing	18	2	18	2	100%
Documentation	3	24	6	23	100%
REVIEW II	24	1	24	1	100%