B. Tech Major Project Report

ITPE-40

on

DETECTION OF MALICIOUS MOBILE WEBPAGES

BY

RIA JAIN (11710582)

HARSH MITTAL (11710580)

ROHAN (11710532)

Group No.: 6

Under the Supervision of

Dr. A.K. Singh, Professor

DEPARTMENT OF COMPUTER ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

KURUKSHETRA-136119, HARYANA (INDIA)

April-May, 2021

CERTIFICATE

We hereby certify that the work which is being presented in this B.Tech. Major Project (ITPE-40) report entitled "**Detection of Malicious Mobile Webpages**" in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Engineering** is an authentic record of our own work carried out during a period from January, 2021 to April, 2021 under the supervision of Dr. A.K. Singh, Professor, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of Candidate

RIA JAIN(11710582)

HARSH MITTAL(11710580)

ROHAN(11710532)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date: April 26, 2021

Signature of supervisor:

Dr. A.K. Singh

Professor

TABLE OF CONTENTS

ABSTRACT

Due to large-scale use of mobile phones, it is unsafe for attacks life phishing. These mobile phones offer features in huge amount, therefore, they are used extensively for accessing web or many other online services. Use of these mobile phones is not bounded to calling, Video Chats and messaging apps but the users use it for financial transactions as well. Phishing is basically a fraud practice that aims to get the information of user using the internet by stating that they are not a fraud entity. It has compromised millions of users' data. These attacks are increasing at a very fast pace. Mobile phones have small screen size, so it is very difficult to differentiate between malicious and legitimate websites. There are plenty of mechanisms to detect malicious webpages on desktop but mobile specific webpages differ significantly from desktop. So, existing techniques such as blacklist method, heuristic method etc. do not work for such webpages. Consequently, it has become a dire need to develop a solution which tells which URL is malicious. Therefore, Aim of this project is to create an android application which will differentiate between malicious and legitimate websites. This project is divided into four stages, firstly it checks if the entered URL is already in the database or not and if it is present in the database it will answer accordingly. If the URL is not present in the database it will check whether domain id is IP address or not and if it is IP address it will warn the user. If it is not the IP address then it will extract features from the entered URL and creates a feature vector. After this, probability is calculated using Naive Bayes and then this probability is added as a feature in the feature vector. Various machine learning algorithms like Support Vector Machine, K Nearest Neighbours, Logistic Regression etc. are then applied to check which algorithm gives the best accuracy.

I. INTRODUCTION

The Internet has now become an important part of everyone's lives. Internet provides all the basic necessities to the users by just clicking a button. According to a report, count of users who have access to the internet has reached 3 billion. And almost one-third of this population use mobile phones [1]. The Internet has made life easier for each person as they can do all their transactions while sitting at home. Nowadays, it is also much easier to access these services because of the number of users using mobile phones and tablets, these gadgets are easy to use and well as portable and they also provide almost all the functionality which PC provides. On the other hand, it has exposed them to various threats like phishing, malware attack etc. These kinds of attacks tempt people into visiting malicious websites and then convince them to enter their personal details like bank details, passwords etc [2]. To save people from these kinds of threats many companies have released anti phishing toolbars. However, according to the reports, there is only one tool which can detect more than 60% of malicious websites [3]. So, it is an urgent need to have a good phishing website detection algorithm.

Mobile phones are increasingly used to access online services. Browsing experience on mobile phones are considerably different because mobile phones screen size is smaller than that of desktop which also affects layout and content of the webpage. Due to the very small screen size of mobile phones, users can only see the part of a URL. So, deceptive words are being placed in the starting of the URL by the creator of a webpage and that URL will be enough to fool the user and get their personal information. Below are some more reasons of why mobile phones are more prone to these attacks:

- Websites can easily be copied by duplicating the source code of legitimate websites
- There are less security mechanisms in mobile phones
- As discussed above, size of mobile phones are usually much smaller than desktop,so it is nearly impossible to detect malicious websites on the basis of their UI.

Phishing is the most common attack among all the security threats. According to Kaspersky Lab, from the first quarter of 2015, an increase of around a million phishing cases have been reported [4]. Currently, lots of malicious website detection systems are there but nobody among them gives the best accuracy when it comes to real-time safe browsing.

For detecting phishing websites,there are many methods which are already applied such as: heuristic techniques, black list method, visual appearance method. In the blacklist method, URL is checked in the list of blacklisted websites, it can be fast for detecting phishing websites but it also has its own drawback as it cannot detect the website which appears only for a day or even for a few hours. Heuristic technique is more efficient than the blacklist method as it can detect zero day attack as well.

Aim of this project is to build an android application which will detect phishing websites with good accuracy. We tried using the least number of features and also the most effective one's which will help in predicting zero day websites. Various ML algorithms will be compared with each other in order to achieve the best accuracy. World wide and country traffic rank is also calculated in this approach. This approach has achieved a higher number of accuracy.

II. MOTIVATION

Over the last decade, the number of cell phone users has increased rapidly at an alarming rate and the number are still growing. Moreover, all the search engines have gradually shifted their focus towards mobile. Similarly, many websites and advertisers progressively shifted to the mobile user's behaviour.

According to Data Breach Investigations Report of Verizon's 2019, almost 32% of the data breach attacks involved phishing activity. Furthermore, "phishing was present in 78% of Cyber-Espionage incidents and the installation and use of backdoors."[5]

According to the Phishing report from Kaspersky Labs in 2019, brazil was the targeted company that year as compared to other countries. This is measured by the share of users whose Anti-Phishing solutions were triggered by users in those countries. The next most targeted country, Australia, jumped up six slots to second place with 17.20% in the same time period [5].

Even in this pandemic situation of our country, amongst the fears of COVID-19 pandemic, the use of malicious websites has increased rapidly. Attackers have created a Coronavirus tracker map to spread malware that is basically focused on stealing the information from the user's phone. A phishing website was created, 'coronavirus-map.com' that seemed to be a benign live tracking map for COVID-19 virus. After identifying this attack, the Health Sector Cybersecurity Coordination Centre reported that the attackers were imitating Johns Hopkins University, a reputed health institution, to contaminate visitors with the AZORult trojan. As mobile phones have limited security in the browser that's why mobile phones are getting infected in a greater amount [6].

All the attackers main focus was mobile phones because mobile phones have less security as compared to desktop PCs because of the fact that mobile phones have less storage, a smaller screen size, different functionalities, etc. So, there are some ways to solve this issue that is based upon an existing static analysis mechanism and we have added some mobile specific functionality and improved existing mechanisms to detect malicious phishing URLs on mobile phones with greater accuracy and security.

III. LITERATURE SURVEY

Number of mobile phone users is increasing day by day. As per report released in 2010, Android phone has now become the prime platform. Over 32.9 million android phones were sold at that time [7]. Various facilities are provided by mobile phones such as online banking systems, online stores etc. According to the survey, at least once in a day, 40% users enter passwords into their phones [8]. Due to this, attackers are shifting their focus from desktop to mobile phone attacks.

Currently, there are lots of solution available to detect the malicious websites and some of them are whitelisting or blacklisting, heuristic, proactive phishing URL detection based methods.

Whitelisting or blacklisting method is efficient for detecting phishing websites. In this method, either a list of malicious webpages or legitimate web pages is created [9]. Whenever a user visits the website, it checks the URL in the existing database and responds accordingly. However, this method also has a drawback as it fails to detect new phishing websites.

DNS based approach [10], is the popular approach for detecting malicious websites. In this approach, the IP address of a phishing website will be validated. The URL will be sent to DNS which then verifies whether the IP address is present in the list of IPs of genuine sites. The site will be considered as a malicious one if the IP address is not present in the list. When using this method there are some problems like exorbitant communication cost, unable to work properly if DNS poisoning is there, increases burden on DNS.

Phishing attack detection using proactive prevention uses genuine existing URLs to generate all possible combinations of URLs [11]. After this, it is checked if each and every combination exists on the web or not. Then the URLs are separated into two sets of phishing and non-phishing. Intensive computation is required for this approach. We need high communication and high computation cost to generate such large no. of combinations and detect them on a regular basis.

Visual similarity based method is basically to compare the visual appearance of the websites. Even though websites look similar in their appearance but there are significant differences

between them. We can compare favicons of the legitimate website and copied website using image processing techniques and if they match beyond a certain threshold value then that URL is a malicious URL [12]. This method requires a large number of resources as well as storage to match the visual similarity.

Heuristic based technique uses static features of the website for classification of malicious and non malicious web pages. But, this method does not work appropriately in many cases. This approach faces performance issues as it takes too much time to compute the result.

IV. PROPOSED APPROACH

Our aim is to classify the URL whether it is malicious or not. Proposed approach is divided into four stages.

Stage I :

System has its own database in which all the URLs which have already been checked are stored. So, to minimize wasting of time as well as resource consuming of the system, entered URL will be first checked into the existing database and if it is present in the database it will respond accordingly and hence avoiding unnecessary computation of the URL. This stage contributes in the fast detection of phishing URLs

Stage II :

As a symbol of identity, non-malicious websites use their company name as their domain name. On the other hand, attackers mostly use their IP address as their domain name in order to avoid the cost of buying domain name. So, after checking the URL in the database, URLs domain name will be extracted and if it is an IP address then the system will warn the user that this site might be a malicious website and hence giving choice to the user to either keep browsing or stop it. And if it is not the IP address system will move to the next stage.

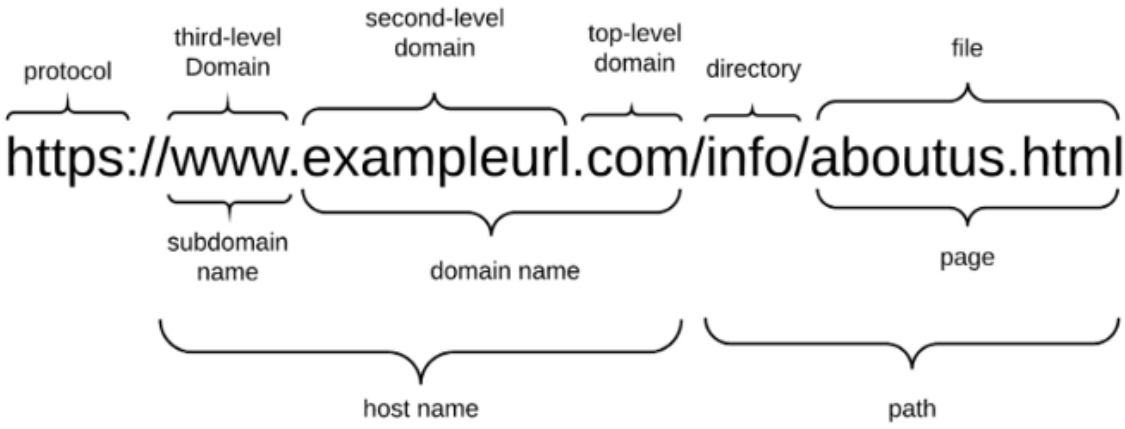

Fig 1. Elements of an URL

Stage III:

In the third stage, the URL will be sent to the server. It will then collect certain features from the URL which will help in deciding whether a site is malicious or not. Along with URL based features, many other types of features have been used in our approach and it includes HTML features, Traffic rank, malicious and non-malicious probability etc.

1. **URL based features:** URL is the first thing whose features should be extracted to classify whether a website is malicious or not. Many URL features helps the ML model to predict the result and some of them are total length of the URL, Number of digits in the URL, number of slash etc. Many malicious websites use IP address of their machines as their domain name. They also include many sub-domains in their URL which increases the length of the URL. Therefore, length of URL, sub-domain is present or not are taken as features. Many other URL features are also included in the feature set like number of semicolons, ampersand symbol, equal sign etc.

2. **Traffic rank:** Traffic rank is basically the ranking of your website versus all the other websites on the internet. We calculated World Traffic rank and Country Traffic rank for the entered URL. Malicious websites are usually less popular so traffic rank work as a major feature in determining whether a website is malicious or not.

3. **HTML features**: HTML is used for making static websites. Certain HTML features are important for predicting whether a website is malicious or not. Some of them are images, white space percentages, location of elements in the web page etc. In total, 8 HTML features are extracted from the website and then they are added in the feature vector. It has been detected that websites with higher number of redirections tend to be a malicious website as it prevents them from being identified by DNS.

4. **Mobile specific features**: There are 4 mobile specific features which have been extracted to detect whether a website is malicious or not and this includes No. of tel API calls, No. of apk API calls, No. of sms API calls, No. of mms API calls. According to the research, malicious web pages put links of inappropriate content in iframes. These web pages also have drive-by-downloads and clickjacking [13].

5. **Javascript features:** Javascript is used to add functionality to the static web pages. Javascript features are extracted because, according to the reports, eval function of javascript is used by almost 6805 websites among 44.4% of top websites from alexa [13]. We check if the webpage has noscript content and then count the number of noscript. Attackers add noscript in the code because it ensures good experience. Number of external and internal javascript is also calculated. Malicious websites will have less number of external javascript. So, in total 8 javascript features are extracted from the webpage and then they are added to the feature set.

Table 1. Different Features in our dataset

Feature Group	Features	Total
Mobile Specific features	No. of sms, No. of tel, No. of mms, No. of apk API calls	4
Javascript features	Presence of JS, Presence of noscript, Presence of internal JS and external Js. No. of JS, No. noscript, No. internal JS and external JS	8
HTML features	Presence of images, Presence of links,Presence of iframes and redirects. No. of images, No. of links, No. of iframes and redirects	8
URL features	Length of the URL, No. of forward slashes, No. question marks, No.of dots, hyphens, underscores, equal signs, ampersand, semicolon and digits	10
Traffic rank	World-wide traffic rank, Country traffic rank	2
Probability	Non-malicious and malicious probability	2

Various features have been extracted from the webpage and the URL and then unnecessary ones are removed from the feature set to get the best accuracy from the ML model.

Stage IV:

This is the final stage of our system. In this stage, URL will be added to the system's database for future use. And the result that whether web page is malicious or not will be displayed to the user.

V. DATA FLOW DIAGRAM

Level 0 DFD

Below figure shows the Level 0 DFD for detection of malicious mobile web pages using our android application:

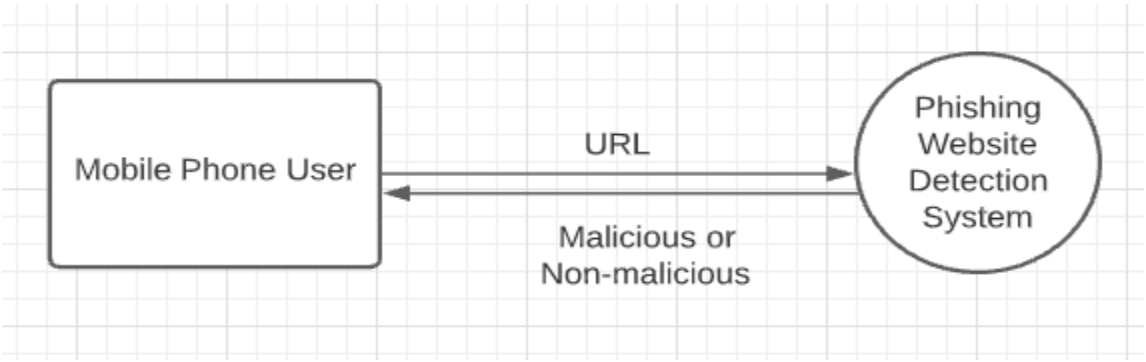

Fig 2. Level 0 DFD for malicious web page detection

Level 1 DFD

Figure 3. represents the different processes involved in the project.

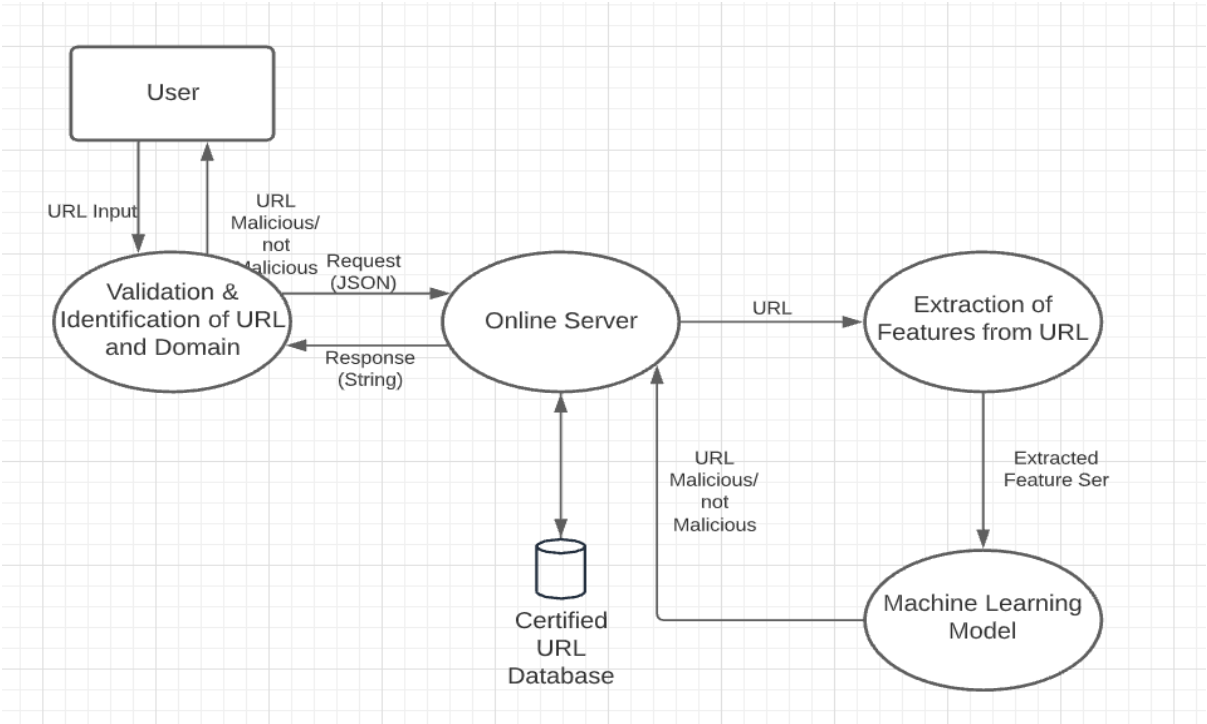

Fig 3. Level 1 DFD for malicious web page detection

Level 2 DFD

- **Feature Extraction:** The fig. 4 represents the various feature groups that are extracted

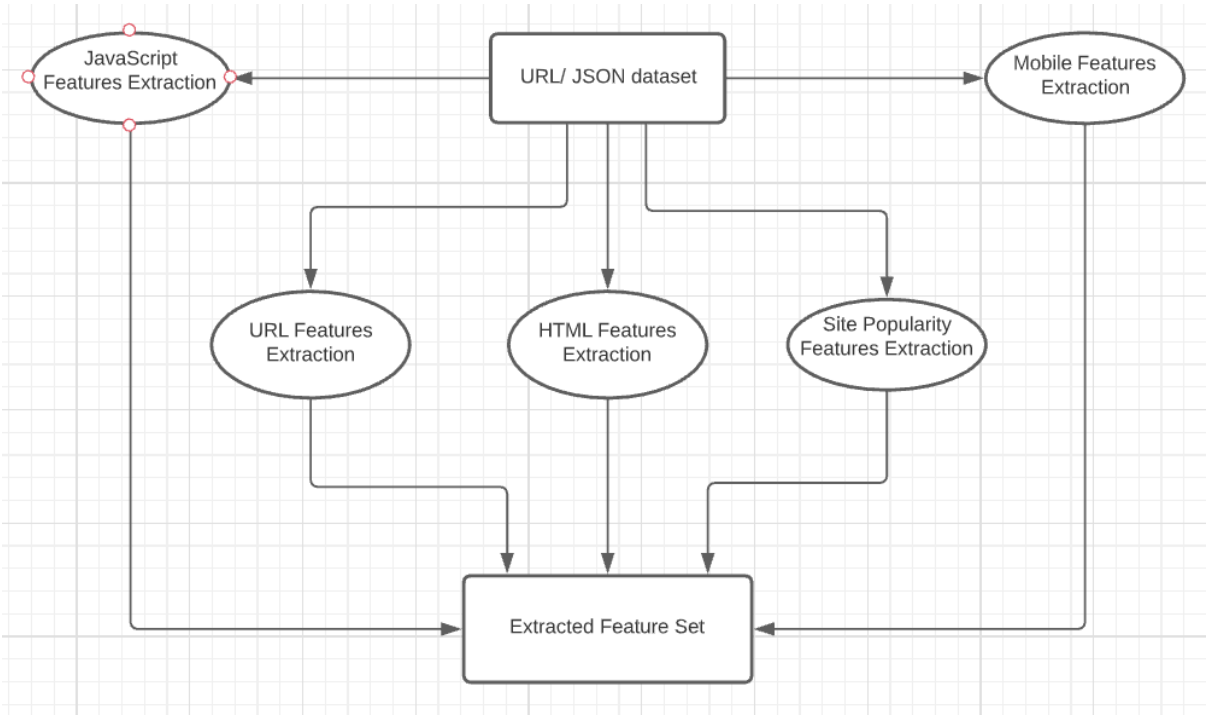

Fig 4. Level 2 DFD for Feature Extraction Process

- **Machine Learning Model:** Below figure 5 shows various steps taken to create a ML model for the backend.

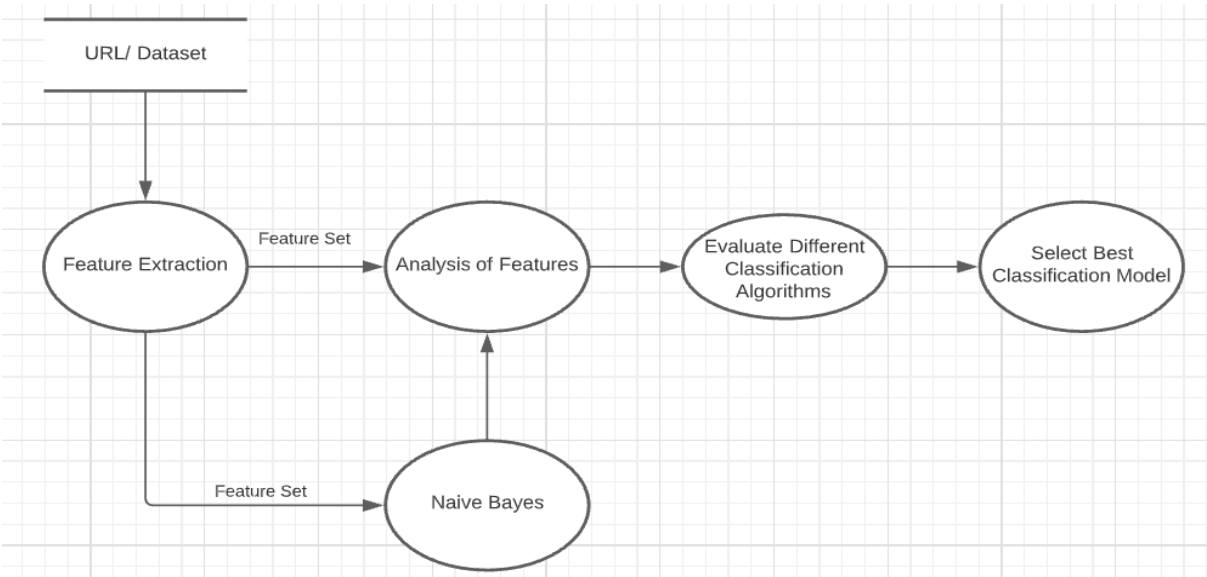

Fig 5. Level 2 DFD for Machine Learning Model

VI. IMPLEMENTATION

The architecture of our project is as follows: Firstly, we have our UI in react native and then there is a server made in flask which is used to integrate frontend with backend. Then finally there is a ML model which will give the result that whether the URL sent by the user is malicious or not.

Data Collection:

To train and test the model, a dataset is needed. So, a list of malicious and non-malicious web pages is maintained and using that list ML model is being trained. Web pages which have different URLs in desktop and mobile browser are included in mobile specific webpages. Below table shows how to identify that URL is mobile specific

Table 2. Mobile specific URL indicators

Top Level Domain	.mobi
Sub Domain	m.,mobile.,touch.,3g.,sp.,s.,mini.,mobileweb.,t.
URL Path Prefix	/mobile, /mobileweb, /m, /mobi, /?m=1, /mobil, /m_home

Feature Extraction and selection:

As discussed above, after collecting the data, various types of features will be extracted from the URL as well as by crawling the web page. It includes mobile specific features, HTML features, javascript features, Traffic Rank, URL features. Extracted features are then added to the feature set. Malicious and Non- malicious probability is also calculated and have added them as a feature in the feature set. For the feature selection purpose, we used the feature importance method which is present in python. After extracting all the features, only some of the significant features that give the best accuracy have been considered as the part of the feature set. Not all the features strongly indicate site maliciousness so some of the features are then dropped by hit and trial methods. Total 21 features of each URL have been considered as the strong indicators for site maliciousness.

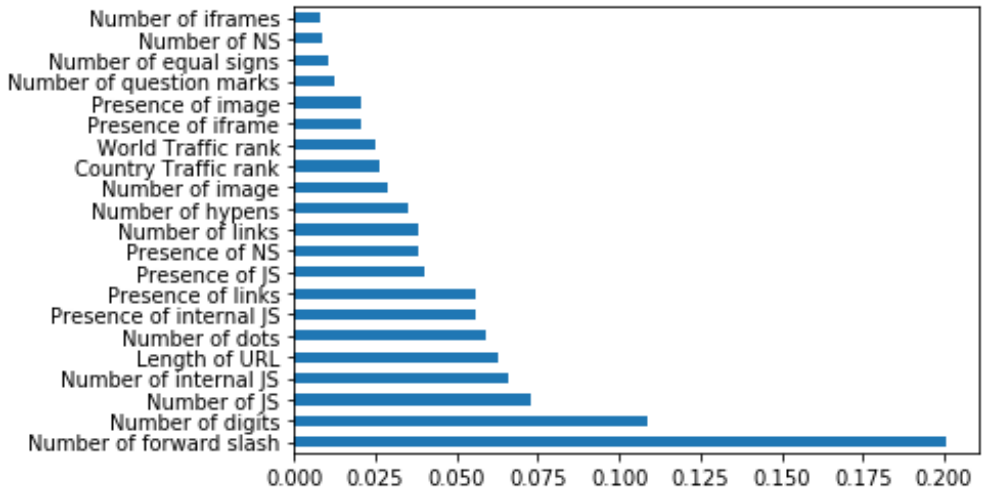

Fig 6. Feature importance graph

Machine Learning Model Selection:

After extracting features of the webpage, the machine learning model has to be trained on that data. To get the best accuracy, various machine learning algorithms have been tried and it includes Decision Tree Classifier, Support Vector Machine, K-Nearest Neighbours, Random Forest Classifier. Among all the algorithms, Random Forest Classifier gives the highest percentage of accuracy.

Flask API:

Now that our model is ready, it should receive data from the user. For this, we have created a server using flask. To expose the web server running on our machine to the internet we used ngrok. Server will receive the json object from the client side and after extracting the url from this json object it has been sent to the machine learning model where features of the given URL will be extracted and it will be classified using the saved machine learning model. Machine learning model is saved using the pickle library of python. Flask server will receive the result from the machine learning model as a json object and it will sent it to the client side.

Application:

An Android application will basically be a client in which the user will enter the URL. We have created our android application in react native. Home screen of the application includes a text area and a button. In the text area, the user will enter the url which he/she wants to check and then he/she will click on the button to check if it is malicious or not. By hitting the button, Flask API will be called. Now as discussed above, the server will send the url to the machine learning model and will also receive the result from the model as a json object and then the server will send the result to the client side. And finally the application will display whether the webpage is malicious or not.

VII. RESULTS AND OBSERVATIONS

Feature set has been created for the training purpose and it is divided into training and testing dataset. We used 25% of the dataset for testing purpose and rest for training purpose. Different classification algorithms have been applied on the test set like Support Vector Machine, K-Nearest Neighbour, Decision Tree Classifier , Random Forest Classifier. Accuracy score has been calculated for each classifier and we observed that Random Forest Classifier gives the best result among all others. Below figure shows the exact accuracy score for each classification algorithm:

```
Accuracy of K Nearest Neighbors algorithm :  0.963855421686747
Accuracy of Support Vector Machine algorithm :  0.963855421686747
Accuracy of Decision Tree algorithm :  0.9759036144578314
Accuracy of Random Forest Classification algorithm :  0.9879518072289156
```

Limitations

The very first drawback of our system is that if the domain id of entered URL is an IP address then our system will alert the user and will not check the legitimacy of the URL. Another is that, phishers can copy the features which are strong indicators for legitimacy detection and hence our system may not give accurate results. According to the current growth of internet users it can be said that no. of phishing attacks will definitely increase in near future and attackers may find a way till then to prove webpage to be benign. To avoid this, our system must be updated every time in order to have less phishing attacks.

Comparison with existing approaches

As we studied, very less amount of work has been done for mobile specific phishing web pages detection. There are two existing approaches, one is KAYO and other one is CANTINA. KAYO is a static analysis technique and it uses static features of websites for classification purpose. This approach does have good accuracy but it also considers all of the features for analysis purpose and not all of them help to indicate whether a website is malicious or not. CANTINA uses TF-IDF for detection of malicious web pages. TF-IDF is basically an information retrieval algorithm which is used for classification. It suffers from performance because it depends on Google safe browsing. So it can be seen that, our approach overcomes

all of the above drawbacks and also gives better accuracy than KAYO and CANTINA. Below table shows the comparison between different features among existing and our approach.

Table 3. Comparison Chart

Factor	**CANTINA**	**KAYO**	**Our approach**
Language	Supports only English	Supports all language	Supports all language
TPR	Around 97%	Around 89%	Around100%
FPR	6%	8%	3.6%
Limitation	Depends on Google safe browsing	Does not drop unnecessary features from the feature set	If the domain id of the URL is an IP address then it just alerts the user.

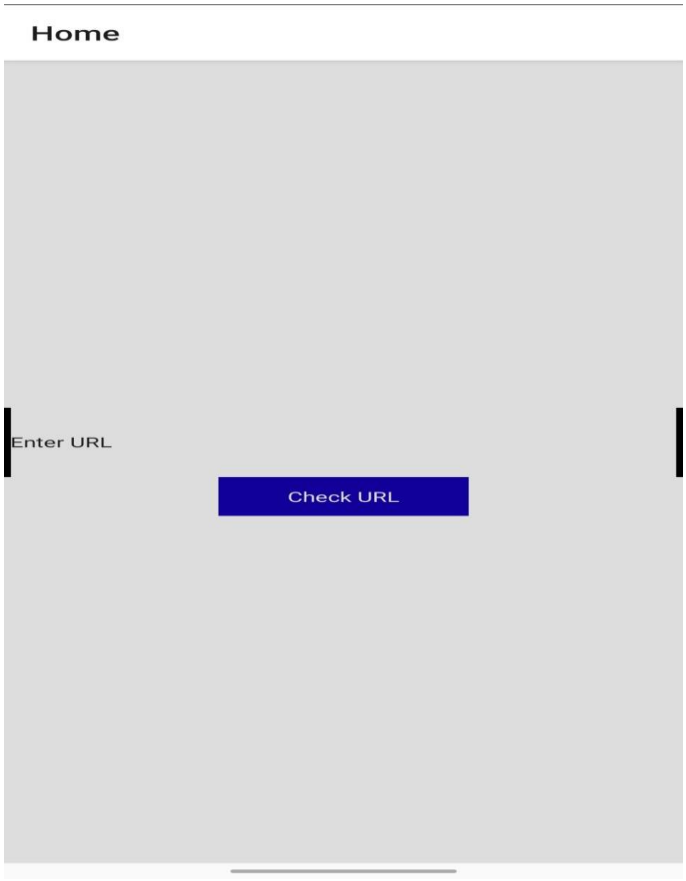

Fig 7.

Above figure is the home page of our android application

Fig 8.

User will enter the URL after erasing the Enter URL text of the text area.

Fig 9.

This is the result screen and whether the site is malicious or non-malicious it will displayed here.

VIII. CONCLUSION AND FUTURE PLAN

The Internet has now become an important part of everyone's lives. For every single thing, the internet is needed be it food order, video calling, browsing etc. And in the current scenario, mobile phone users are increasing day by day therefore it is an urgent need to have a system which classifies whether a website is malicious or not. But systems who detect mobile specific malicious web pages are less as compared to desktop sites. Mobile web pages and desktop sites have huge differences in their appearance, content, layout etc. Present systems for detection of mobile specific web pages have their own drawbacks as some are not cost effective others are not for the websites who appear for just one day or may be for one week. So our approach overcomes all of the above drawbacks with a very high accuracy rate. We look forward to enhancing our system so that it will be convenient for the users. We would like to aware the audience about the phishing attacks and hence using our phishing detection system for their own safety.

REFERENCES

[1] Yousif, H., Al-saedi, K. H., & Al-Hassani, M. D. (2019). Mobile Phishing Websites Detection and Prevention Using Data Mining Techniques. International Journal of Interactive Mobile Technologies, 13(10).

[2]Jun, Ho., Huh, & Hyoungshick, Kim.: Phishing Detection with Popular Search Engines: Simple and Effective. Available at: https://link.springer. com/chapter/10.1007/978-3-642 -27901-0_15

[3] Yue, Zhang, Jason Hong & Lorrie Cranor.: CANTINA: A Content-Based Approach to Detecting Phishing Web Sites. Available at: https://www.cs.cmu.edu/~jasonh/publications/www2007-cantina-final.pdf

[4] Shcherbakova, T., Vergelis, M., & Demidova, N. (2015). Spam and phishing in Q2 2015. Quarterly Spam Reports. Available at: https://securelist.com/spam-and-phishing-in-q2-of-2015/71759/ (Last accessed on 26 May 2020)

[5] Crane C. (2019). 20 Phishing Statistics to Keep You from Getting Hooked in 2019. Available at: https://www.thesslstore.com/blog/20-phishing-statistics-to-keep-you-from -getting -hooked-in-2019/ (Last accessed on 26 May 2020)

[6] Crane C. (2020). Coronavirus Scams: Phishing Websites & Emails Target Unsuspecting Users. Available at https://www.thesslstore.com /blog/coronavirus -scams-phishing- websites -emails-target-unsuspecting-users/ (Last accessed on 26 May 2020)

[7] Canalys. Google's Android becomes the world's leading smart phone platform. Canalys research release 2011/013, 2011.

[8] Adrienne Porter Felt & David Wagner.: Phishing on Mobile Devices. Available at: https://people.eecs.berkeley.edu/~daw/papers/mobphish-w2sp11.pdf

[9] Li, L., Berki, E., Helenius, M., & Ovaska, S. (2014). Towards a contingency approach with whitelist-and blacklist-based anti-phishing applications: what do usability tests indicate?. Behaviour & Information Technology, 33(11), 1136-1147.

[10] Gastellier-Prevost, S., Granadillo, G. G., & Laurent, M. (2011, February). A dual approach to detect pharming attacks at the client-side. In 2011 4th IFIP International Conference on New Technologies, Mobility and Security (pp. 1-5). IEEE.

[11] Ferolin, R. J., & Kang, C. U. (2012). Phishing attack detection, classification and proactive prevention using fuzzy logic and data mining algorithm.

[12] Rushikesh Josh. (2015). Interactive Phishing Filter. Available at https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1426&context=etd_projects

[13] Amrutkar, C., Kim, Y. S., & Traynor, P. (2016). Detecting mobile malicious webpages in real time. IEEE Transactions on Mobile Computing, 16(8), 2184-2197.

APPENDIX:

COMPLETE CONTRIBUTORY SOURCE CODE

A. Code for ML Model

```
import pandas as pd
import requests
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt
from math import pi
from math import sqrt
from math import exp
from bs4 import BeautifulSoup
from sklearn.utils.optimize import _check_optimize_result
from functools import partial
from sklearn.gaussian_process import GaussianProcessRegressor

#list containing mobile specific websites
listOf_Malicious_Websites=set()

#function to extract only mobile specific websites from feed.txt
def mobile_specific_websites():
    y=set()
    file1=open('feed.txt','r')
    y=file1.readlines()
    for lines in y:
        result=False
        url=lines.strip()
        y1=url.split('/')
        y2=list()
        for iter in y1:
            y3=iter.split('.')
            for j in y3:
                y2.append(j)
        #print(y2)
        for iter in y2:
            if (iter=='mobi' or iter=='m' or iter=='mobile' or
iter=="touch" or iter=='3g' or iter=='sp' or iter=='s' or iter=='mini'
or iter=='mobileweb' or iter=='t' or iter=='?m=1'or iter=='mobil' or
iter=='m_home' ):
                result=True
        #print(result)
        if(result==True):
```

```
            listOf_Malicious_Websites.add(url)
        #print(listOf_Malicious_Websites)
    count_malicious=len(listOf_Malicious_Websites)
    print("Total number of mobile specific malicious sites are : ",
count_malicious)

#this list will contain the features of all URL
features_Of_Url=list()

#function to extract features from mobile specific websites
def malicious_website_feature_extraction():
    for URL in listOf_Malicious_Websites:
        try :
            response = requests.get(URL)
        except :
            continue
        temporary_list=list()
        soup = BeautifulSoup(response.content, 'html5lib')
        #print(soup)
        flag_js=0
        count_js=0
        for row in soup.findAll('script'):
            count_js=count_js+1
        if count_js != 0 :
            flag_js=1

        flag_noScript=0
        count_noScript=0
        for row in soup.findAll('noscript'):
            count_noScript=count_noScript+1
        if count_noScript != 0 :
            flag_noScript=1

        flag_external_js=0
        flag_internal_js=0
        count_external_js=0
        for row in soup.findAll('script',attrs='src'):
            count_external_js=count_external_js+1
        count_internal_js=count_js-count_external_js
        if count_external_js != 0 :
            flag_external_js=1
        if count_internal_js != 0 :
            flag_internal_js=1
```

```
        flag_image=0
        count_image=0
        for row in soup.findAll('img'):
            count_image=count_image+1
        if count_image != 0 :
            flag_image=1

        flag_iframe=0
        count_iframe=0
        for row in soup.findAll('iframe'):
            count_iframe=count_iframe+1
        if count_iframe != 0 :
            flag_iframe=1

        flag_redirect=0
        count_redirect=0
        for row in soup.findAll('meta', attrs={'http-equiv':'Refresh'
}):
            count_redirect=count_redirect+1
        if count_redirect != 0 :
            flag_redirect=1

        flag_links=0
        temp=list()
        for row in soup.findAll('a'):
            temp.append(row.get('href'))
        count_links=len(temp)
        if count_links != 0 :
            flag_links=1

        temporary_list.append(flag_js)
        temporary_list.append(flag_noScript)
        temporary_list.append(flag_external_js)
        temporary_list.append(flag_internal_js)
        temporary_list.append(flag_image)
        temporary_list.append(flag_iframe)
        temporary_list.append(flag_redirect)
        temporary_list.append(flag_links)

        temporary_list.append(count_js)
        temporary_list.append(count_noScript)
        temporary_list.append(count_external_js)
```

```
        temporary_list.append(count_internal_js)
        temporary_list.append(count_image)
        temporary_list.append(count_iframe)
        temporary_list.append(count_redirect)
        temporary_list.append(count_links)
        count_sms=0
        count_tel=0
        count_apk=0
        count_mms=0
        for s in temp:
            if(isinstance(s,str)):
                if(s.startswith('sms')):
                    count_sms=count_sms+1
                if(s.startswith('tel')):
                    count_tel=count_tel+1
                if(s.endswith('apk')):
                    count_apk=count_apk+1
                if(s.startswith('mms')):
                    count_mms=count_mms+1
        temporary_list.append(count_sms)
        temporary_list.append(count_tel)
        temporary_list.append(count_apk)
        temporary_list.append(count_mms)
        url_len= len(URL)
        #print(url_len)
        temporary_list.append(url_len)
        number_forward_slash=0
        number_que_mark=0
        number_dots=0
        number_hyphen=0
        number_underscore=0
        number_equals=0
        number_ampersand=0
        number_semi_colon=0
        number_digit=0
        for iter in URL:
            if iter == '/' :
                number_forward_slash = number_forward_slash+1
            if iter =='?' :
                number_que_mark = number_que_mark+1
            if iter =='.' :
                number_dots = number_dots+1
            if iter =='-' :
```

```
                number_hyphen = number_hyphen+1
            if iter =='_' :
                number_underscore = number_underscore+1
            if iter =='=' :
                number_equals = number_equals+1
            if iter =='&' :
                number_ampersand = number_ampersand+1
            if iter ==';' :
                number_semi_colon = number_semi_colon+1
            if iter =='0' or iter =='1' or iter =='2' or iter=='3' or
iter =='4' or iter =='5' or iter =='6' or iter =='7' or iter =='8' or
iter =='9' :
                number_digit=number_digit+1
        temporary_list.append(number_forward_slash)
        temporary_list.append(number_que_mark)
        temporary_list.append(number_dots)
        temporary_list.append(number_hyphen)
        temporary_list.append(number_underscore)
        temporary_list.append(number_equals)
        temporary_list.append(number_ampersand)
        temporary_list.append(number_semi_colon)
        temporary_list.append(number_digit)
        try:
            traffic_rank =
int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/dat
a?cli=10&dat=s&url=" + URL).read(), "xml").find("REACH")["RANK"])
        except:
            traffic_rank=0
        temporary_list.append(traffic_rank)
        try:

counttry_traffic_rank=int(bs4.BeautifulSoup(urllib.request.urlopen("htt
p://data.alexa.com/data?cli=10&dat=s&url=" + URL).read(),
"xml").find("COUNTRY")["RANK"])
        except:
            country_traffic_rank=0
        temporary_list.append(country_traffic_rank)
        temporary_list.append(1)
        features_Of_Url.append(temporary_list)
    print("mobile done")

f2=open('legitimate_sites.txt','r')
urls=f2.readlines()
```

```
def benign_website_feature_extraction():
    for URL in urls:
        try :
            response = requests.get(URL)
        except :
            continue
        temporary_list=list()
        soup = BeautifulSoup(response.content, 'html5lib')
        #print(soup)
        flag_js=0
        count_js=0
        for row in soup.findAll('script'):
            count_js=count_js+1
        if count_js != 0 :
            flag_js=1

        flag_noScript=0
        count_noScript=0
        for row in soup.findAll('noscript'):
            count_noScript=count_noScript+1
        if count_noScript != 0 :
            flag_noScript=1

        flag_external_js=0
        flag_internal_js=0
        count_external_js=0
        for row in soup.findAll('script',attrs='src'):
            count_external_js=count_external_js+1
        count_internal_js=count_js-count_external_js
        if count_external_js != 0 :
            flag_external_js=1
        if count_internal_js != 0 :
            flag_internal_js=1

        flag_image=0
        count_image=0
        for row in soup.findAll('img'):
            count_image=count_image+1
        if count_image != 0 :
            flag_image=1

        flag_iframe=0
        count_iframe=0
```

```
        for row in soup.findAll('iframe'):
            count_iframe=count_iframe+1
        if count_iframe != 0 :
            flag_iframe=1

        flag_redirect=0
        count_redirect=0
        for row in soup.findAll('meta', attrs={'http-equiv':'Refresh'
}):
            count_redirect=count_redirect+1
        if count_redirect != 0 :
            flag_redirect=1

        flag_links=0
        temp=list()
        for row in soup.findAll('a'):
            temp.append(row.get('href'))
        count_links=len(temp)
        if count_links != 0 :
            flag_links=1

        temporary_list.append(flag_js)
        temporary_list.append(flag_noScript)
        temporary_list.append(flag_external_js)
        temporary_list.append(flag_internal_js)
        temporary_list.append(flag_image)
        temporary_list.append(flag_iframe)
        temporary_list.append(flag_redirect)
        temporary_list.append(flag_links)

        temporary_list.append(count_js)
        temporary_list.append(count_noScript)
        temporary_list.append(count_external_js)
        temporary_list.append(count_internal_js)
        temporary_list.append(count_image)
        temporary_list.append(count_iframe)
        temporary_list.append(count_redirect)
        temporary_list.append(count_links)
        count_sms=0
        count_tel=0
        count_apk=0
        count_mms=0
        for s in temp:
```

```
            if(isinstance(s,str)):
                if(s.startswith('sms')):
                    count_sms=count_sms+1
                if(s.startswith('tel')):
                    count_tel=count_tel+1
                if(s.endswith('apk')):
                    count_apk=count_apk+1
                if(s.startswith('mms')):
                    count_mms=count_mms+1
        temporary_list.append(count_sms)
        temporary_list.append(count_tel)
        temporary_list.append(count_apk)
        temporary_list.append(count_mms)
        url_len= len(URL)
        temporary_list.append(url_len)
        number_forward_slash=0
        number_que_mark=0
        number_dots=0
        number_hyphen=0
        number_underscore=0
        number_equals=0
        number_ampersand=0
        number_semi_colon=0
        number_digit=0
        for iter in URL:
            if iter == '/' :
                number_forward_slash=number_forward_slash+1
            if iter =='?' :
                number_que_mark=number_que_mark+1
            if iter =='.' :
                number_dots=number_dots+1
            if iter =='-' :
                number_hyphen=number_hyphen+1
            if iter =='_' :
                number_underscore=number_underscore+1
            if iter =='=' :
                number_equals=number_equals+1
            if iter =='&' :
                number_ampersand=number_ampersand+1
            if iter ==';' :
                number_semi_colon=number_semi_colon+1
```

```
            if iter =='0' or iter =='1' or iter =='2' or iter =='3' or
iter =='4' or iter =='5' or iter =='6' or iter =='7' or iter =='8' or
iter =='9' :
                number_digit=number_digit+1
        temporary_list.append(number_forward_slash)
        temporary_list.append(number_que_mark)
        temporary_list.append(number_dots)
        temporary_list.append(number_hyphen)
        temporary_list.append(number_underscore)
        temporary_list.append(number_equals)
        temporary_list.append(number_ampersand)
        temporary_list.append(number_semi_colon)
        temporary_list.append(number_digit)
        try:

traffic_rank=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.
alexa.com/data?cli=10&dat=s&url=" + URL).read(),
"xml").find("REACH")["RANK"])
        except:
            traffic_rank=0
        temporary_list.append(traffic_rank)
        try:

country_traffic_rank=int(bs4.BeautifulSoup(urllib.request.urlopen("http
://data.alexa.com/data?cli=10&dat=s&url=" + URL).read(),
"xml").find("COUNTRY")["RANK"])
        except:
            country_traffic_rank=0
        temporary_list.append(country_traffic_rank)
        temporary_list.append(0)
        features_Of_Url.append(temporary_list)

#Naive Bayes as feature implementation
# To split the dataset according to class value
def split(dataset):
    split = dict()
    for iter in range(len(dataset)):
        feature_vector = dataset[iter]
        value_cls = feature_vector[-1]
        if (value_cls not in split):
            split[value_cls] = list()
        split[value_cls].append(feature_vector)
    return split
```

```
# For mean calculation
def mean_numbers(numbers):
    a = sum(numbers)
    b = len(numbers)
    return a/float(b)

# standard deviation calculation
def standard_deviation_numbers(numbers):
    average_numbers = mean_numbers(numbers)
    length = len(numbers) - 1
    a = sum([(it - average_numbers)**2 for it in numbers])
    variance = a/float(length)
    return sqrt(variance)

def summarize(dataset):
    smrize = [(mean_numbers(column),
standard_deviation_numbers(column), len(column)) for column in
zip(*dataset)]
    del(smrize[-1])
    return smrize

# statistics calculation
def summarize_by_class(dataset):
    separated = split(dataset)
    summaries = dict()
    for value_cls, rows in separated.items():
        summaries[value_cls] = summarize(rows)
    return summaries

# Gaussian probability calculation
def calculate_probability(it, mean, standard_deviation):
    # print("standard_deviation value",standard_deviation)
    if(standard_deviation==0):
      standard_deviation = 0.01
    a = exp(-((it - mean)**2
    b = (2 * standard_deviation**2 )))
    expo = a/b
    try:
        a = sqrt(2*pi)
        b = a*standard_deviation
        return (1/b) * expo
    except:
```

```
        return 1

# class probability calculation
def calculate_class_probabilities(summaries, row):
    count_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] =
summaries[class_value][0][2]/float(count_rows)
        for iter in range(len(class_summaries)):
            mean, standard_deviation, _ = class_summaries[iter]
            probabilities[class_value] *=
calculate_probability(row[iter], mean, standard_deviation)
    return probabilities


if __name__ == "__main__":
    mobile_specific_websites()
    malicious_website_feature_extraction()
    benign_website_feature_extraction()
    #df=pd.DataFrame(features_Of_Url, columns = ['Presence of
JavaScript' ,'Presence of NoScript' ,'Presence of external JavaScript'
,'Presence of internal JavaScript' ,'Presence of image' ,'Presence of
iframe','Presence of redirects' ,'Presence of links' ,'No. of
JavaScript' ,'No. of NoScript' ,'No. of external JavaScript' ,'No. of
internal JavaScript' ,'No. of image' ,'No. of iframes' ,'No. of
redirects' ,'No. of links' ,'No. of sms API call','No. of tel API
call','No. of apk API call','No. of mms API call','Length of the
URL','No. of forward slash','No. of question marks', 'No. of dots',
'No. of hyphens' , 'No. of underscore', 'No. of equal signs', 'No. of
ampersand' , 'No. of semi-colon','No. of digits','World Traffic
rank','Country Traffic rank', 'Output'])
    data_frame=pd.DataFrame(features_Of_Url, columns = ['Presence of
JavaScript' ,'Presence of NoScript' ,'Presence of external JavaScript'
,'Presence of internal JavaScript' ,'Presence of image' ,'Presence of
iframe','Presence of redirects' ,'Presence of links' ,'No. of
JavaScript' ,'No. of NoScript' ,'No. of external JavaScript' ,'No. of
internal JavaScript' ,'No. of image' ,'No. of iframes' ,'No. of
redirects' ,'No. of links' ,'No. of sms API call','No. of tel API
call','No. of apk API call','No. of mms API call','Length of the
URL','No. of forward slash','No. of question marks', 'No. of dots',
'No. of hyphens' , 'No. of underscore', 'No. of equal signs', 'No. of
```

```
ampersand' , 'No. of semi-colon','No. of digits','World Traffic
rank','Country Traffic rank', 'Output'])
    arr=data_frame.to_numpy()

    #Calculating class probabilities
    temporary0=list()
    temporary1=list()
    summaries = summarize_by_class(arr)
    for iter in range(len(data_frame)) :
        probability = calculate_class_probabilities(summaries,
arr[iter])
        temporary0.append(probability[0])
        temporary1.append(probability[1])

    #inserting the calculated benign and malicious probability
    data_frame1 = pd.DataFrame(arr)
    data_frame1.insert(32, 'benign_probability', temporary0)
    data_frame1.insert(33, 'malicious_probability', temporary1)

    #dropping unnecessary features by hit and trial method
    final_dataFrame = data_frame1.drop(['malicious_probability',
'benign_probability'],axis=1)
    final_dataFrame.columns
    final_X = final_dataFrame.iloc[:,:-1].values
    final_Y = final_dataFrame.iloc[:,-1].values

    #splitting the dataset into training and test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, Y_train, Y_test =
train_test_split(final_X,final_Y, test_size = 0.25, random_state=1)


    #Applying K Nearest Neighbors Algorithm
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import confusion_matrix
    from sklearn import metrics
    KNN = KNeighborsClassifier(n_neighbors = 13, metric = 'minkowski' ,
p=2)
    KNN.fit(X_train,Y_train)
    y_prediction_KNN = KNN.predict(X_test)

    #Accuracy calculation of K nearest neighbour algorithm
    cm_KNN = confusion_matrix(Y_test,y_prediction_KNN)
```

```
    accuracy_KNN=metrics.accuracy_score(Y_test,y_prediction_KNN)
    print("Accuracy of K Nearest Neighbors algorithm : ",accuracy_KNN)

    #Applying Support Vector Machine
    from sklearn.svm import SVC
    SVM = SVC(kernel='rbf',random_state=1)
    SVM.fit(X_train,Y_train)
    y_prediction_SVM = SVM.predict(X_test)

    #Accuracy calculation of Support Vector Machine algorithm
    cm_SVM = confusion_matrix(Y_test,y_prediction_SVM)
    accuracy_SVM=metrics.accuracy_score(Y_test,y_prediction_SVM)
    print("Accuracy of Support Vector Machine algorithm :
",accuracy_SVM)

    #Applying Decision Tree Classifier
    from sklearn.tree import DecisionTreeClassifier
    DT = DecisionTreeClassifier(criterion='entropy',random_state=1)
    DT.fit(X_train,Y_train)
    y_prediction_DT = DT.predict(X_test)

    #Accuracy calculation of Decision Tree Classification algorithm
    cm_DT = confusion_matrix(Y_test,y_prediction_DT)
    accuracy_DT=metrics.accuracy_score(Y_test,y_prediction_DT)
    print("Accuracy of Decision Tree algorithm : ",accuracy_DT)

    #Applying Random Forest Classifier
    from sklearn.ensemble import RandomForestClassifier
    RFC = RandomForestClassifier(n_estimators =
50,criterion='entropy',random_state=0) #yaha pe random forest 0/1 check
karke dekhlio
    RFC.fit(X_train,Y_train)
    y_prediction_RFC = RFC.predict(X_test)

    #Accuracy calculation of K nearest neighbour algorithm
    cm_RFC = confusion_matrix(Y_test,y_prediction_RFC)
    accuracy_RFC = metrics.accuracy_score(Y_test,y_prediction_RFC)
    print("Accuracy of Random Forest Classification algorithm :
",accuracy_RFC)

    import pickle
    pickle.dump(RFC,open('saved_model_2.pkl','wb'),protocol=2)
```

B. Code for Classifying the URL

```
import requests
from bs4 import BeautifulSoup
import urllib,bs4
import urllib.request
import numpy as np
import pickle
def retrive_pred(URL):
  try :
    r = requests.get(URL)
  except :
    return "Error"
  temporary_list=list()
  soup = BeautifulSoup(r.content, 'html.parser')
  flag_js=0
  count_js=0
  for row in soup.findAll('script'):
      count_js=count_js+1
  if count_js != 0 :
      flag_js=1

  flag_noScript=0
  count_noScript=0
  for row in soup.findAll('noscript'):
      count_noScript=count_noScript+1
  if count_noScript != 0 :
      flag_noScript=1

  flag_external_js=0
  flag_internal_js=0
  count_external_js=0
  for row in soup.findAll('script',attrs='src'):
      count_external_js=count_external_js+1
  count_internal_js=count_js-count_external_js
  if count_external_js != 0 :
      flag_external_js=1
  if count_internal_js != 0 :
      flag_internal_js=1

  flag_image=0
  c_img=0
  for row in soup.findAll('img'):
```

```
        c_img=c_img+1
    if c_img != 0 :
        flag_image=1

    flag_iframe=0
    count_iframe=0
    for row in soup.findAll('iframe'):
        count_iframe=count_iframe+1
    if count_iframe != 0 :
        flag_iframe=1

    flag_redirect=0
    count_redirect=0
    for row in soup.findAll('meta', attrs={'http-equiv':'Refresh' }):
        count_redirect=count_redirect+1
    if count_redirect != 0 :
        flag_redirect=1

    flag_links=0
    temp=list()
    for row in soup.findAll('a'):
        temp.append(row.get('href'))
    count_links=len(temp)
    if count_links != 0 :
        flag_links=1

    temporary_list.append(flag_js)
    temporary_list.append(flag_noScript)
    temporary_list.append(flag_external_js)
    temporary_list.append(flag_internal_js)
    temporary_list.append(flag_image)
    temporary_list.append(flag_iframe)
    temporary_list.append(flag_redirect)
    temporary_list.append(flag_links)

    temporary_list.append(count_js)
    temporary_list.append(count_noScript)
    temporary_list.append(count_external_js)
    temporary_list.append(count_internal_js)
    temporary_list.append(c_img)
    temporary_list.append(count_iframe)
    temporary_list.append(count_redirect)
    temporary_list.append(count_links)
```

```
count_sms=0
count_tel=0
count_apk=0
count_mms=0
for s in temp:
    if(isinstance(s,str)):
        if(s.startswith('sms')):
            count_sms=count_sms+1
        if(s.startswith('tel')):
            count_tel=count_tel+1
        if(s.endswith('apk')):
            count_apk=count_apk+1
        if(s.startswith('mms')):
            count_mms=count_mms+1
temporary_list.append(count_sms)
temporary_list.append(count_tel)
temporary_list.append(count_apk)
temporary_list.append(count_mms)
url_len= len(URL)
#print(url_len)
temporary_list.append(url_len)
number_forward_slash=0
number_que_mark=0
number_dots=0
number_hyphen=0
number_underscore=0
number_equals=0
number_ampersand=0
number_semi_colon=0
number_digit=0
for i in URL:
    if i== '/' :
        number_forward_slash=number_forward_slash+1
    if i=='?' :
        number_que_mark=number_que_mark+1
    if i=='.' :
        number_dots=number_dots+1
    if i=='-' :
        number_hyphen=number_hyphen+1
    if i=='_' :
        number_underscore=number_underscore+1
    if i=='=' :
        number_equals=number_equals+1
```

```
      if i=='&' :
          number_ampersand=number_ampersand+1
      if i==';' :
          number_semi_colon=number_semi_colon+1
      if i=='0' or i=='1' or i=='2' or i=='3' or i=='4' or i=='5' or
i=='6' or i=='7' or i=='8' or i=='9' :
          number_digit=number_digit+1
  temporary_list.append(number_forward_slash)
  temporary_list.append(number_que_mark)
  temporary_list.append(number_dots)
  temporary_list.append(number_hyphen)
  temporary_list.append(number_underscore)
  temporary_list.append(number_equals)
  temporary_list.append(number_ampersand)
  temporary_list.append(number_semi_colon)
  temporary_list.append(number_digit)
  try:

traffic_rank=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.
alexa.com/data?cli=10&dat=s&url=" + URL).read(),
"xml").find("REACH")["RANK"])
  except:
    traffic_rank=0
  temporary_list.append(traffic_rank)
  try:

country_traffic_rank=int(bs4.BeautifulSoup(urllib.request.urlopen("http
://data.alexa.com/data?cli=10&dat=s&url=" + URL).read(),
"xml").find("COUNTRY")["RANK"])
  except:
    country_traffic_rank=0
  temporary_list.append(country_traffic_rank)
  print(temporary_list)
  model=pickle.load(open('saved_model.pkl','rb'))
  result = np.array2string(model.predict([temporary_list]))
  if result == '[0]':
    return "Non malicious"
  else:
    return "Malicious"

#if __name__ == "__main__":
 # print(retrive_pred('https://webflow.com/blog/business-website-
examples'))
```

C. Code for creating a server

```
import json
import pickle
from flask import Flask
from flask import request
import predict
app  = Flask(__name__)

@app.route('/abc', methods=['POST', 'GET'])
def hello_world():
    print('checking')
    url = request.json
    data_set = predict.retrive_pred(url['url'])
    print(data_set)
    json_dump = json.dumps(data_set)
    print(request.data)
    #url = request.json
    print(url['url'])
    return json_dump

if __name__ == '__main__':
    app.run()
```

D. Code for user interface using React Native

```
import React, { Component } from 'react';
import { StyleSheet, Text, SafeAreaView, TextInput, TouchableOpacity,
View, Button} from 'react-native';
import {NavigationContainer} from '@react-navigation/native';
import {createStackNavigator} from '@react-navigation/stack';

function HomeScreen({navigation}) {

  var [text, onChangeText] = React.useState("Enter URL");
  var [response, changeResponse] = React.useState("processing")
  var onButtonPress = async () => {
  var data = {
      method: 'POST',
      headers: {
```

```
        'Accept': 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        url: text
      })
    }
  console.log(text)
  var res
  await fetch('https://f82b63c2aac9.ngrok.io/abc',data)
  .then((response) => response.json())
  .then((json) => {
    console.log('checkinnnggg')
    console.log(json)
    res = json
    return json;
  })
  .catch((error) => {
    console.error(error);
  });
  console.log(res)
  await changeResponse(res)
  console.log(response)
  navigation.navigate('Result', {response})
}

  return (

    <View style={styles.container}>
        <TextInput
          style={styles.input}
          onChangeText={onChangeText}
          value={text}
        />
        <Button title="Check URL" onPress={onButtonPress}/>

    </View>
  );
}

function ResultScreen({ route, navigation }) {
  const { response } = route.params;
  return (
```

```
    <View style = { styles.container }>
      <Text>{response}</Text>
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}

const Stack = createStackNavigator();

function MyStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Result" component={ResultScreen} />
    </Stack.Navigator>
  );
}

export default function App() {

  return (
    <NavigationContainer>
      <MyStack />
    </NavigationContainer>
  );
}

const styles = StyleSheet.create(
  {
    container: {
      justifyContent: 'center',
      flex: 1,
      margin: 10,
    },

    ActivityNameTextCss: {
      fontSize: 22,
      color: 'black',
      textAlign: 'center',
    },
  });
```