WEB PHISHING DETECTION

A PROJECT REPORT

Submitted by

ANAND.K (950019106003)

GURU MOORTHY.B(950019106009)

MOHAMED MUSTHAQ.I.B(950019106026)

VAISHALI.A(950019106045)

TEAM ID:PNT2022TMID49605

BACHELOR OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING,
ANNA UNIVERSITY REGIONAL CAMPUS TIRUNELVELI

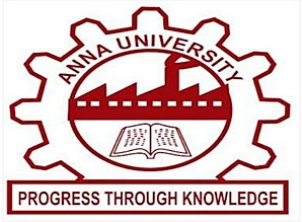


TABLE OF CONTENTS

S.NO

1.INTRODUCTION

1.1 PROJECT OVERVIEW

Now a days Phishing becomes a main area of concern for security researchers because it is not difficult to create the fake website which looks so close to legitimate website. Experts can identify fake websites but not all the users can identify the fake website and such users become the victim of phishing attack.

Phishing attacks are becoming successful because lack of user awareness. Since phishing attack exploits the weaknesses found in users, it is very difficult to mitigate them but it is very important to enhance phishing detection techniques.

1.2 PURPOSE

The aim of the phishers is to acquire critical information like username, password and bank account details. Cyber security persons are now looking for trustworthy and steady detection techniques for phishing websites detection. Machine learning technology is used for detection of phishing URLs by extracting and analysing various features of legitimate and phishing URLs. Decision Tree, random forest and K-Nearest neighbour algorithms are used to detect phishing websites. The aim of the project is to detect phishing and legitimate URLs using machine learning algorithm.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

S. NO	AUTHOR	TITLE	NAME OF JOURNAL
1.	Xiao, X., Zhang, D., Hu, G., Jiang, Y. & Xia	CNN–MHSA: A Convolutional Neural Network and multi-head self-attention combined approach for detecting phishing websites	Neural Networks, Volume 125, May 2020

- In the above CNN–MHSA model, several layers including the convolutional layer and the MHSA layer were organically connected, where the former was used for learning URLs' features while the latter was aimed to calculate the corresponding features' weights.

S. NO	AUTHOR	TITLE	NAME OF JOURNAL
2.	FaanZhengQiaoYanVictor C.M.LeungF.Richard YuZhongMing	Highway deep pyramid convolution neural network combining word-level and character-level representations for phishing website detection	Computers & Security Volume 114, March 2022, 102584

- Highway deep pyramid convolution neural network first receives the URL string sequences as input and then performs character-level embedding and word-level embedding.
- Then, it uses the Highway network to connect the character-level embedding representation and word-level embedding representation of the URL and extracts local features of different sizes from the region embedding layer.
- Finally, it passes them into the designed deep pyramid structure network to capture the global representation of the URL.

S. NO	AUTHOR	TITLE	NAME OF JOURNAL
3.	Youness Mourtaji, Mohammed Bouhorma, Daniyal Alghazzawi, Ghadah Aldabbagh, and Abdullah Alghamdi	Hybrid Rule-Based Solution for Phishing URL Detection Using Convolutional Neural Network	Wireless Communications and Mobile ComputingArticle ID 8241104(2021)

- The above study presented a novel hybrid solution based on fuzzy logical rules as a new technique of detecting phishing URLs, where static analysis was used as base treatment: visual comparison or features extraction depending on the URL domain name was already blacklisted for the domains which contained the legitimate domain names database.
- In other complicated cases, the process was undertaken through the dynamic analysis by analysing the behaviors and the content of the web page belonging to this URL.

2.2 REFERENCES

[1]https://doi.org/10.1016/j.neunet.2020.02.013

[2] https://doi.org/10.1016/j.cose.2021.102584

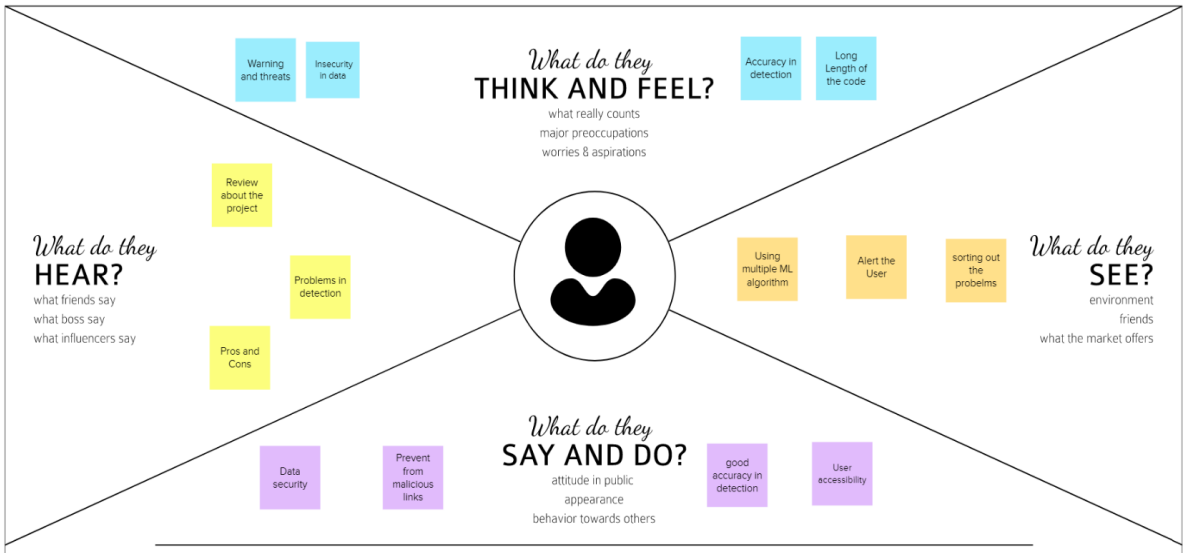
[3] https://doi.org/10.1155/2021/8241104

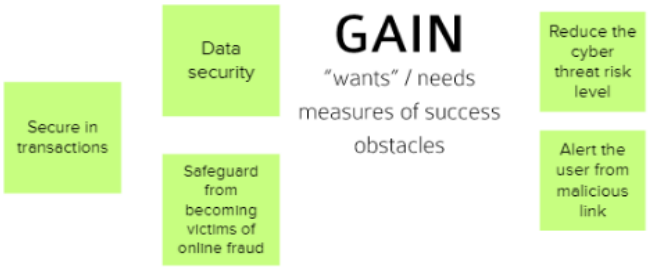
2.3 PROBLEM STATEMENT DEFINITION

Many users unwittingly click phishing domains every day and every hour. The attackers are targeting both the users and the companies. This leads to loss of personal data, credentials, medical records etc. The model is developed to detect the phishing websites and to protect our data.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS





3.2 IDEATION AND BRAINSTORMING

1

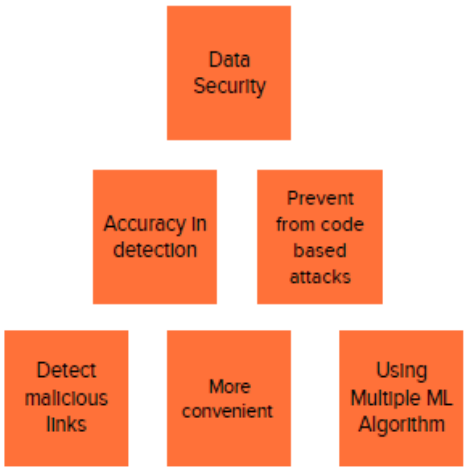
Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

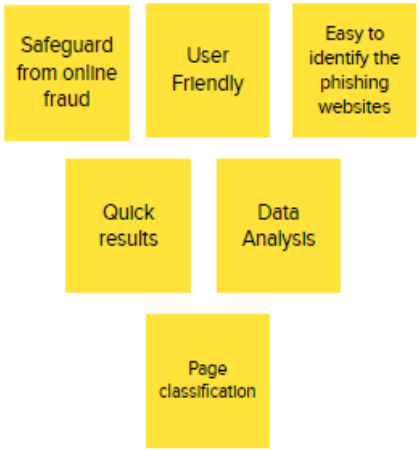
5 minutes

During the Covid-19 crisis, everything went through online even classes and work became online but there was a problem with security. there are several phishing websites ruled over the internet. Around 75% of organisations around the world experienced some kind of phishing attack in 2020. The average cost of a data breach is $3.92M.The top 3 types of data that are compromised in phishing attacks are
I)Credentials
II)Personal data
III)Medical reports

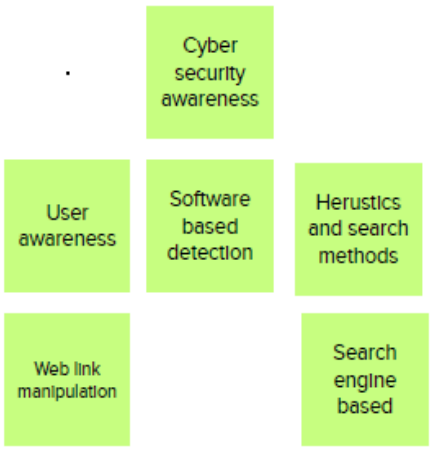
Anand.K



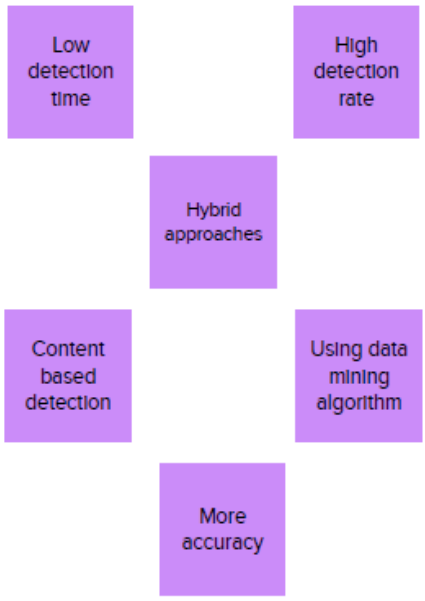
Guru moorthy.B

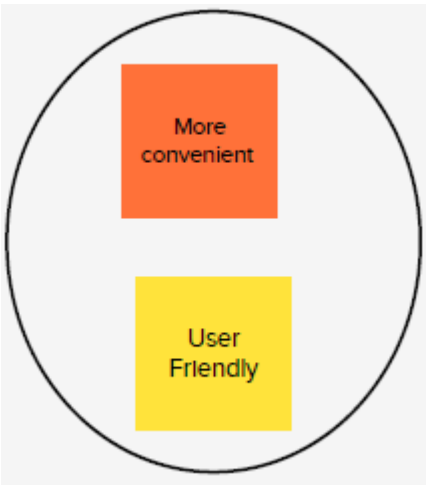


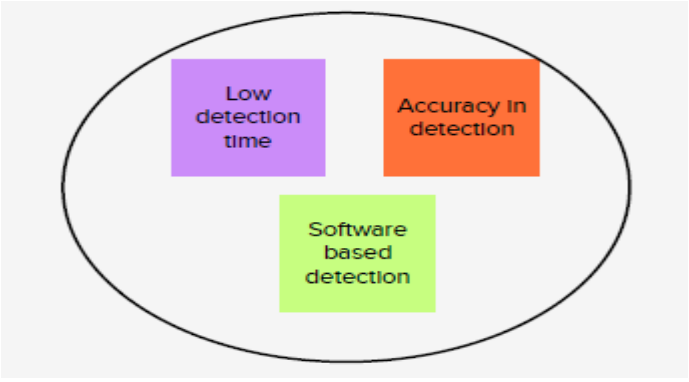
Mohamed Musthaq.I.B

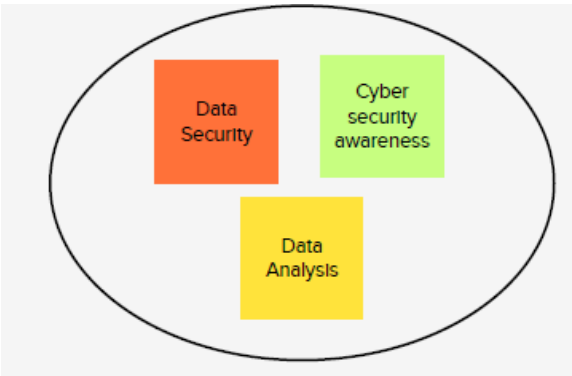


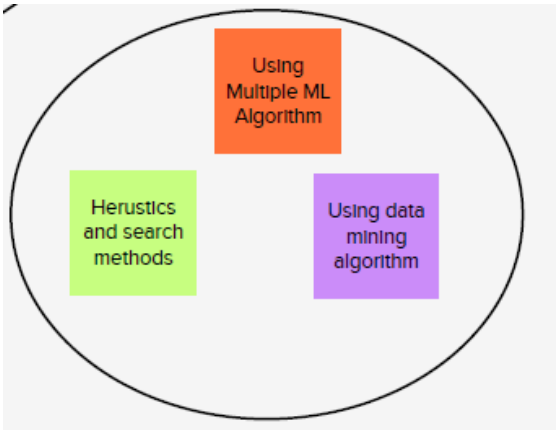
Vaishali.A

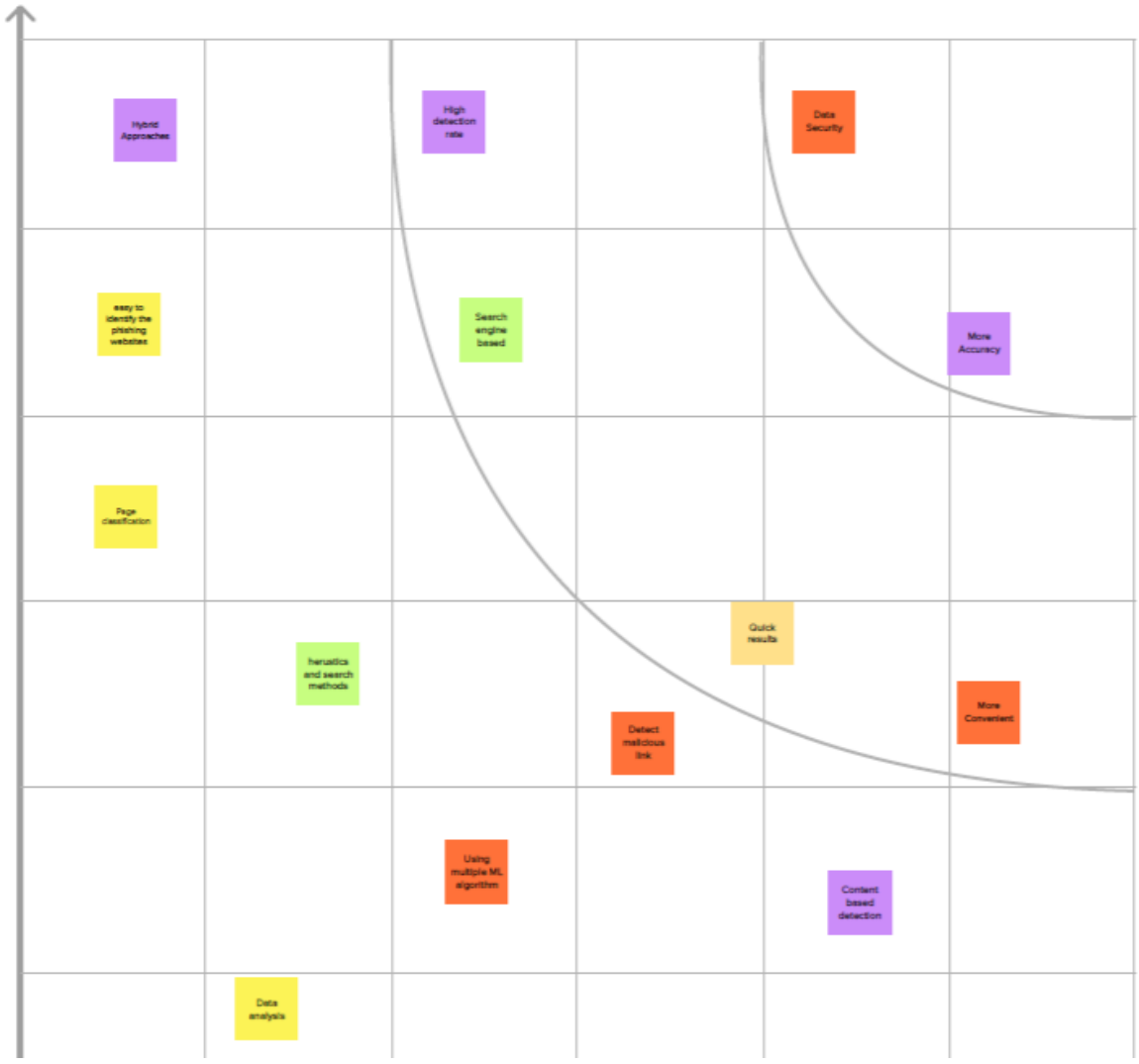







Data Security
Cyber security awareness
Data Analysis


Using Multiple ML Algorithm
Herustics and search methods
Using data mining algorithm


Hybrid Approaches
High detection rate
Data Security
easy to identify the phishing websites
Search engine based
More Accuracy
Page classification
Quick results
Herustics and search methods
More Convenient
Detect malicious link
Using multiple ML algorithm
Content based detection
Data analysis

3.3 PROPOSED SOLUTION

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Many users unwittingly click phishing domains every day and every hour. The attackers are targeting both the users and the companies. This leads to loss of personal data, credentials, medical records etc.
2.	Idea / Solution description	1)Machine learning is a powerful tool used to strive against phishing attacks. The website uses Random Forest algorithm.
3.	Novelty / Uniqueness	The website uses random forest algorithm.
4.	Social Impact / Customer Satisfaction	1)It prevents the user from the duplicate websites and loss of data, money etc. 2) Many cybercrimes must be stopped using this website and privacy must be secured
5.	Business Model (Revenue Model)	This website has premium subscription which include a chrome extension so the detection of phishing website will be easy to the users.(available in future)
6.	Scalability of the Solution	1)The website can handle large number of users simultaneously and provides accurate solution. 2)The detection of phishing websites is quick and easy to use.

3.4 PROBLEM SOLUTION FIT

Project Title: Web Phishing Detection

Project Design Phase-I - Solution Fit Template

Team ID: PNT2022TMID49605

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

1. Banks and online service providers
2. Credit card companies and social media websites

6. CUSTOMER CONSTRAINTS CC

1)It prevents the user from fraudulent websites.
2) It protects the privacy of the customers and prevents from loss of data or money.

5. AVAILABLE SOLUTIONS AS

1. Black list method which evades the attackers by using a fast flux algorithm.
2. Heuristic-based detection which discovers zero-hour phishing attack

Explore AS, differentiate

Focus on J&P, into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

1)Credit cards and banking details should be prevented from the attackers
2)Data should be protected from hackers.

9. PROBLEM ROOT CAUSE RC

1)It causes problems in banking and money transactions and leads to money loss.
2)It may also affect the medical sector.
3)It also affects social media users due to data insecurity.

7. BEHAVIOUR BE

1. The website has a chatbot where the user can clear their queries.
2. Security Concerns websites differ from all other types of browsing websites

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

1)Good accuracy in detection by using ML Algorithm.
2) URL-based detection techniques use features of the URLs themselves to determine whether the link is malicious

4. EMOTIONS: BEFORE / AFTER EM

BEFORE	AFTER
1) Unable to detect malicious links or phishing URLs. 2) People are not aware of fake websites	1) Aware of malicious links or phishing URLs. 2)Fake links or websites could be found.

10. YOUR SOLUTION SL

1. Detection of phishing URLs by extracting and analyzing various features of legitimate and phishing URLs.
2. The website can provide a more quick and more accurate solution.
3. The website can handle a large number of users simultaneously.

8.CHANNELS of BEHAVIOUR CH

ONLINE:
1)The solution is based on online users through the internet.
OFFLINE:
1)Phishing awareness training that educates end users on specific phishing attacks.

Identify strong TR & EM

4.REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

S.NO	Functional Requirement	Sub Requirement
1	User Input	User gives an URL as input and check whether it is legal or phishing site.
2	Comparison of Websites	The model compares the given website using Blacklist and Whitelist approach.
3	Feature Extraction	If nothing was found on comparison then it extracts feature using heuristic and visual similarity approach.
4	Prediction	The model predicts the URL using Machine Learning algorithm Random Forest which have highest accuracy and the model compare other algorithm such as Logistic Regression, Naive Bayes, decision Tree and K- nearest neighbour.
5	Classifier	The model sends all output to classifier and final result will be produced.
6	Announcement	The model displays whether the website is legal site or phishing site.
7	Accuracy	The model produce the high accuracy of 96%.

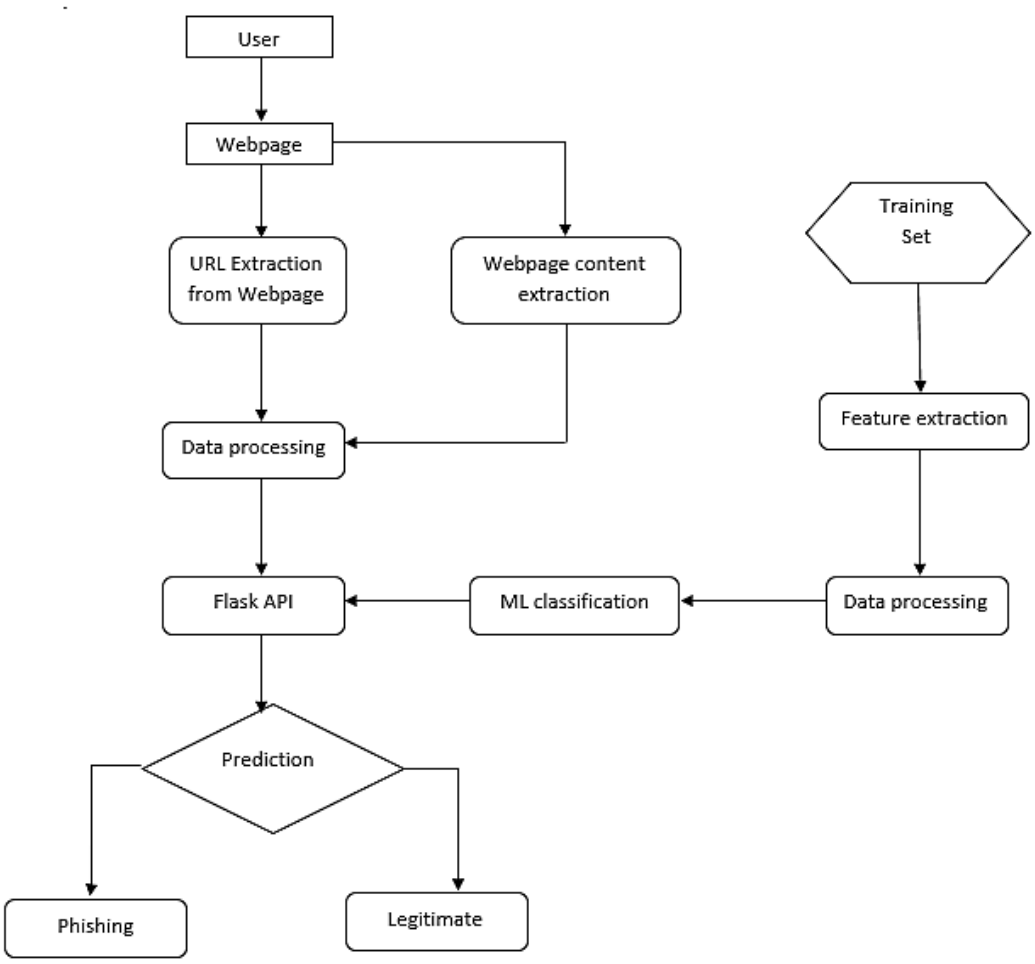
4.2 NON - FUNCTIONAL REQUIREMENTS

S.NO	Non-Functional Requirement	Description
1	Usability	User can access the website easily and it is user-friendly. The website detects the phishing website using the features of the dataset.
2	Security	As the project is based on security, the website is secured to use.
3	Reliability	It is easy to detect and takes less time to detect the phishing website.
4	Performance	The model has good performance since it uses ML algorithm.
5	Availability	The model uses the dataset from phish tank and other sources.
6	Scalability	It can handle large number of users.

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) maps out the flow of information for any process or system. A neat and clear DFD can depict the right amount of the system requirement graphically. It highlights the movement of information as well as the sequence of steps or events required to complete a task.



5.2 SOLUTION & TECHNICAL ARCHITECTURE:

SOLUTION DESCRIPTION:

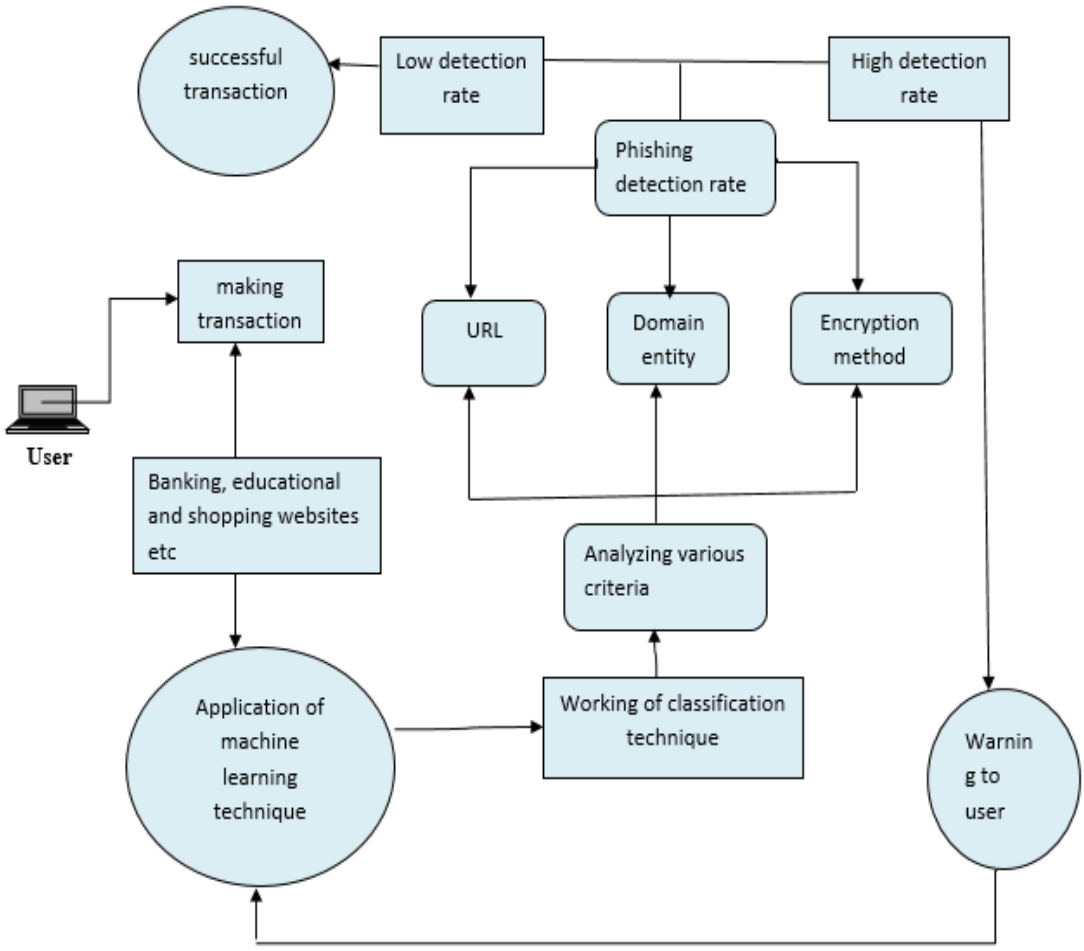
The solution to the problem is, the users who use phishing websites unknowingly which makes their data insecure, this website helps to find the solution to their problem at any time.

The user can simply type website name(phishing website) and check whether the website is original or insecure.

SOLUTION ARCHITECTURE DIAGRAM:

TechnicalArchitecture:

The Deliverable shall include the architectural diagramas below



TechnicalArchitecture:

The Deliverable shall include the architectural diagramas below

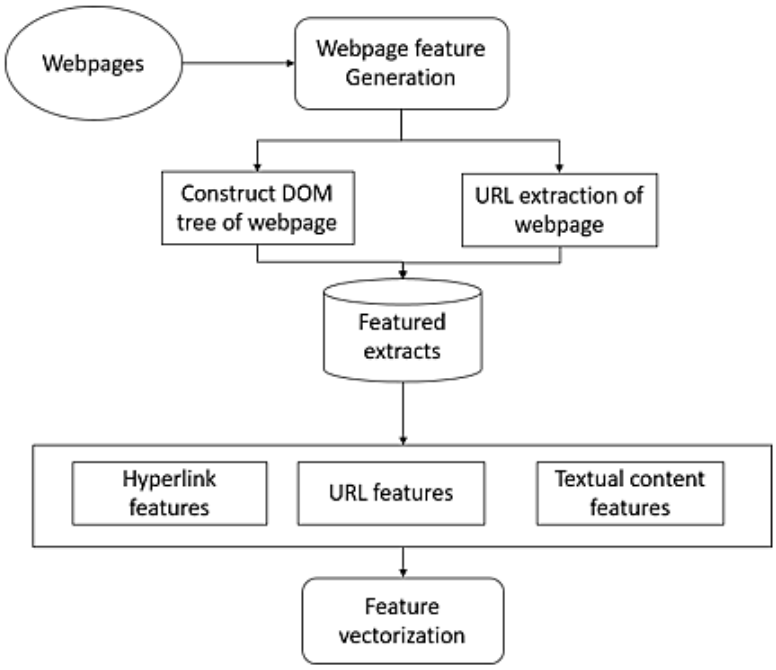


Fig 1: PREPROCESSING PHASE

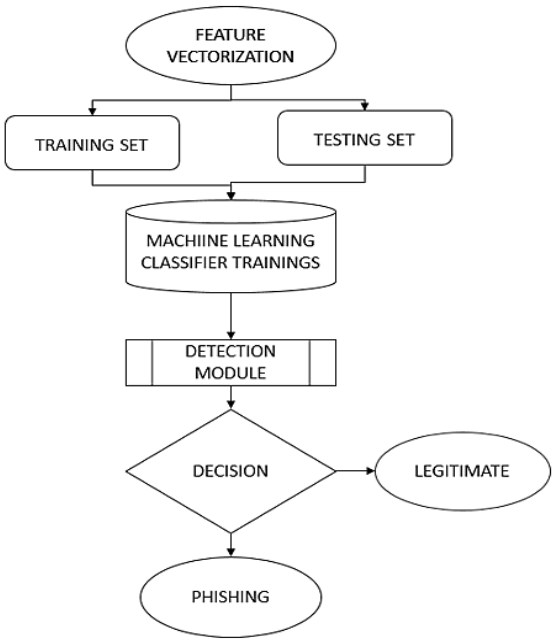


FIG 2: DETECTION PHASE

5.3 USER STORIES

User type	Functional Requirement (Epic)	User story number	User story/ task	Acceptance criteria	Priority	Release
Customer	Open - webpage	USN-1	User enter into the website using the web URL.		High	Sprint-1
Customer (Web user)	User input(website)	USN-1	User gives the input URL in the required field and waits for validation.	Accessing the website without any problem.	High	Sprint-1
Customer Care Executive	Feature extraction	USN-1	After comparison, in case if none found on comparison then the feature is extracted using heuristic and visual similarity approach.	Comparison between websites for security.	High	Sprint-1
Administrator	Prediction	USN-1	Here, the Model predicts the URL websites using Random Forest classifier.	Suitable model is predicted using ML algorithms.	High	Sprint-1

	Classifier	USN-2	The model sends the output to classifier in order to produce final result.	Suitable classifier is identified for producing the result.	Medi um	Sprint-2

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

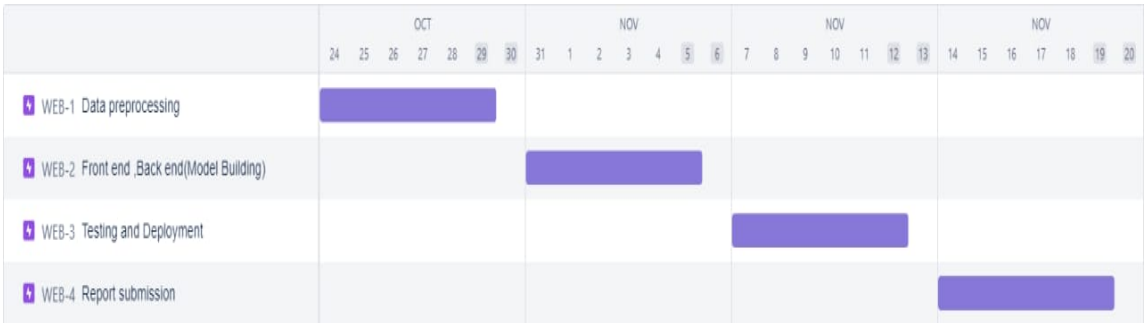
SPRINT PLANNING

Sprint	Functional Requirement(EPIC)	User story number	User story/Task	Story Points	Priority	Team Members
Sprint1	Data pre-processing	USN-1	Pre-processing the dataset	1	High	All team members.
Sprint2	Front and back end (Model building)	USN-2	Developing a front end and back end	2	Medium	All team members.
Sprint3	Testing and deployment	USN-3	Deployment of phising detection	1	High	All team members.
Sprint4	Report submission	USN-4	Overall completion of project	2	High	All team members.

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Total story points	Duration	Sprint start date	Sprint end date(planned)	Story points completed(as on planned end date)	Sprint release date(actual)
Sprint1	20	6days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint2	20	6days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint3	20	6days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint4	20	6days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

6.3 REPORTS FROM JIRA



7. CODING & SOLUTIONING

7.1 FEATURE 1

The pre - processed dataset is taken from IBM website. The dataset is created by using 11055 URLs which has both phishing and legitimate websites. A total of 30 features is selected to build the model.

A URL is defined as follows:

<Scheme >:< scheme-specific-part >

The scheme represents the resource's access mechanism or network protocol communication such as HTTP, FTP,etc,. The scheme selected defines the rest of the URL. For the HTTP protocol, a URL can be

<http> :// < host >:< port > / < URL_path >

It begins with the domain name or IP address of the host. At that point, there is a port number to associate it with and the URL way that gives subtle elements on how the resource can be accessed to (e.g., http://host.com:80/page).

LEXICAL AND HOST BASED METHOD FEATURES:

1. having_IPhaving_IP_Address: This feature evaluates whether the current URL uses an IP address instead of a standard domain name. In our system, if the URL uses an IP address, the feature having_IPhaving_IP_Address"" will have the value -1or it will get 1.

2.URLURL_Length: The malicious files uses large URL. If the length of the URL is larger than 75 characters "URLURL_Length," it will get the value -1. Otherwise, if the length is between 54 and 75 characters, it will have the value 0(0 means suspicious) or 1.

3.Shortining_Service: The shortening services avoids the number of characters limited by some websites. It can be used to redirect a user to another website. In this dataset, value -1 is assigned for a URL that comes from a shortening service, or else 1.

4.having_At_Symbol: The "@" in the path of a URL is used to redirect what is on its right position.In this dataset, if the URL uses the "@" symbol will have the -1, or else 1.

6.URL_of_anchor: In the URL of the webpage, redirection to another webpage is represented by the <a> tag. The number of <a> tags found in the source code was examined. It was further analysed that if the hyperlinks that were found in this tag are different from the domain name, then it check the URL of anchor<31% returns 1, or URL of anchor ≥ 31% and

≤ 67% returns 0, otherwise returns –1.

7.Links_in_tag: In HTML (HyperText Markup Language), it is probable to add some extra information to the web page by <meta> tags, or some client codes with <script> tags, or <link> to add some extra resources to the target web pages. So, like Anchor tag the domain name in the URL is less than 25%, “links_in_tag” will get the value 1, or else if it is between 25% and 8181%, then it will return 0 , or else return -1.

8.Abnormal_URL: Legitimate websites generally use the same identity in the domain name and URL in the request; the identity can be gathered using WHOIS information. So the phishers try to hide their identity. So, if that is the case, “abnormal_url” that will return -1, if host name is in URL means it return 1.

9.Submitting_to_email: It describes the fact that phishers send information filled in forms by emails using “email:to” option in <form> tag, which is very suspicious. If it uses “email:to” option it will return –1 or else it return 1.

CONTENT - BASED METHOD FEATURES:

10.double_slash_redirecting: The legitimate websites make redirections before arriving at the URL in question, but phishing can use the redirection for more than three times. So, it was deduced that if the redirection was made one time, the feature “double_slash_redirecting” would be marked as safe and return 1; or else if it was made two or three times, it return 0.

11. on_mouseover: JavaScript handling events such as the mouse and keyboard. The option of “on_MouseOver” can change the status bar of the current web page, this can help phishers to show a fake URL on the status bar. So, if this is the case it return -1 or else it return 1.

12.RightClick: If the website does not permit right click option, then it will return -1 or else it return 1.

13.Iframe: An iframe can be represented as a website inside a website. iframes can have the possibility to be hidden. If the website using iframe means it return –1 or else it return 1.

SECURITY AND IDENTITY METHOD FEATURES:

14.age_of_domain: Legitimate websites purchase domain names for at least one year or sometimes more, but for phishers who aim to execute their attacks for a short period, their

domain age is approximately six months. This information is collected from the WHOIS database. So, if the domain name of the URL in the test is less than six months, then "age_of_domain"it will return -1 or else it will return 1.

15.SSLfinal_State: Using HTTP as a protocol of communication between a user and a server, the network is encrypted, but that does not mean that the used digital certificate can be trusted. Even phishers have a trusted digital certificate, its age is almost in the average of a one-year lifetime. Legitimate websites often buy trusted digital certificates for many years. If it returns the response code it will return 1 or else it return -1.

16.DNSRecord: In WHOIS database, it was checked if there are any DNS records for the URL in question, if there is no dns record for domain it return -1 or else it return 1.

17.Google_Index: If the current URL appears in a search engine like Google Index, it will be legitimate and it return 1 or else it return –1.

18.Statistical_report: PhishTank is an organization that specializes in detecting phishing websites that appear daily, so if the domain appears into the PhishTank database, if it present in Phishing IPs means it return –1 or else it return 1.

OTHER FEATURES:

19.Links_pointing_to_page: If the number of links pointing to webpage = 0 returns 1, number of links pointing to webpage > 0 # and ≤ 2 returns 0 it is considered as phishing , otherwise it returns –1.

20.Page_Rank: PageRank is a normalized value from 0 to 1 to measure the importance of a webpage. PageRank less than 0.2 is considered as phishing

21.web_traffic: If the website is ranked among the top 100,000 according to Alexadatabase rank, the URL is suspicious.If website rank < 100,000 returns 1, website rank > 100,000 returns 0, otherwise returns -1 -

22.popUpWidnow: The presence of popup window with the Text field is an indication of phishing or else it is legitimate.

23.SFH: If Server Form Handler (SFH) is empty, blank or refers to a dissimilar domain, the link is phishing or suspicious if it has SFH means it is legitimate.

24.Favicon: If the favicon displayed from the domain is at variant from that in the address

bar, such URL is considered phishing and it return –1, or else it return 1.

25.Prefix_Suffix: Its presence in a URL indicates phishing attackand it return -1. This is usually indicated with the use of dash (-)or else it return 1.

MACHINE LEARNIING ALGORITHMS:

i) LOGISTIC REGRESSION:

Logistic regression is the machine learning algorithm which comes under the classification of supervised learning. It predicts the output based on the given set of dependent variables. The predicted output is the categorical independent variable. The outcome can be either Yes or No, 0 or 1, true or False, etc.

The Equation of Logistic regression can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- The equation of the straight line can be written as:

$$Y= b_0 + b_1x_1 + b_2x_2 +\ldots\ldots\ldots+ b_nx_n$$

- In Logistic Regression Y can be between 0 and 1 only, divide the above equation by (1-y):

$$Y / 1-Y ; 0 \text{ for } Y=0, \text{ and infinity for } Y=1$$

- To get the range between -[infinity] to +[infinity], taking logarithm of the equation it will become:

$$\log(Y / 1-Y) = b_0 + b_1x_1 + b_2x_2 +\ldots\ldots\ldots+ b_nx_n$$

The above equation is the final equation for Logistic Regression.

ii) RANDOM FOREST:

Random Forest is a popular machine learning algorithm which comes under the supervised learning. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. The model contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. The random forest takes the prediction from

each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

iii) K - NEAREST NEIGHBOUR:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. It assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. The algorithm stores all the available data and classifies a new data point based on the similarity. When new data appears then it can be easily classified into a well suite category by using K- NN algorithm. At the training phase, the algorithm just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

iv) DECISION TREE:

Decision tree is a tree-structured classifier, where the internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas

Leaf nodes are the output of those decisions and do not contain any further branches.

To select the best attribute for the root node and for sub-nodes, a technique called Attribute selection measure or ASM is used.. There are two popular techniques for ASM, which are:

- Information Gain
- Gini Index

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a classInformation.

Gain = Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy(s) = -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- S= Total number of samples
- P(yes)= probability of yes
- P(no)= probability of no

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm. Gini index can be calculated using the below formula:

Gini Index= $1- \sum_j P_j^2$

v) NAIVE BAYES:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features. It performs well in Multi-class predictions as

compared to the other Algorithms. The formula for Bayes' theorem is given as:

$$P(A|B) = P(B|A)\ P(A) / P(B)$$

Where,

- P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.
- P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.
- P(A) is Prior Probability: Probability of hypothesis before observing the evidence.
- P(B) is Marginal Probability: Probability of Evidence.

7.2 FEATURE 2:

CODE:

Input Script:

```
import regex
from tldextract import extract
import socket
from bs4 import BeautifulSoup
import urllib.request
import whois
import requests
import favicon
import re
from googlesearch import search


#checking if URL contains any IP address. Returns -1 if contains else returns 1
def having_IPhaving_IP_Address(url):
    match=regex.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])\\/)|'  #IPv4
                '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-
fA-F]{1,2})\\/)'  #IPv4 in hexadecimal
                '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}',url)     #Ipv6
    if match:
      #print match.group()
      return -1
    else:
```

```
        #print 'No matching pattern found'
        return 1

#Checking for the URL length. Returns 1 (Legitimate) if the URL length is less than 54
characters
#Returns 0 if the length is between 54 and 75
#Else returns -1;
def URLURL_Length (url):
    length=len(url)
    if(length<=75):
        if(length<54):
            return 1
        else:
            return 0
    else:
        return -1

#Checking with the shortening URLs.
#Returns -1 if any shortening URLs used.
#Else returns 1
def Shortining_Service (url):

match=regex.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\
.gs|'
                   'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'

'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'

'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
                   'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'

'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'

'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.g
d|tr\.im|link\.zip\.net',url)
    if match:
        return -1
    else:
        return 1
```

```
#Checking for @ symbol. Returns 1 if no @ symbol found. Else returns 0.
def having_At_Symbol(url):
    symbol=regex.findall(r'@',url)
    if(len(symbol)==0):
        return 1
    else:
        return -1

#Checking for Double Slash redirections. Returns -1 if // found. Else returns 1
def double_slash_redirecting(url):
    for i in range(8,len(url)):
        if(url[i]=='/'):

            if(url[i-1]=='/'):
                return -1
    return 1

#Checking for - in Domain. Returns -1 if '-' is found else returns 1.
def Prefix_Suffix(url):
    subDomain, domain, suffix = extract(url)
    if(domain.count('-')):
        return -1
    else:
        return  1

#checking the Subdomain. Returns 1 if the subDomain contains less than 1 '.'
#Returns 0 if the subDomain contains less than 2 '.'
#Returns -1 if the subDomain contains more than 2 '.'
def having_Sub_Domain(url):
    subDomain, domain, suffix = extract(url)
    if(subDomain.count('.')<=2):
        if(subDomain.count('.')<=1):
            return 1
        else:
            return 0
    else:
        return -1

#Checking the SSL. Returns 1 if it returns the respomse code and -1 if exceptions are thrown.
def SSLfinal_State(url):
```

```
    try:
        response = requests.get(url)
        return 1
    except Exception as e:
        return -1

#domains expires on ≤ 1 year returns -1, otherwise returns 1

def Domain_registeration_length(url):
    try:
        domain = whois.whois(url)
        exp=domain.expiration_date[0]
        up=domain.updated_date[0]
        domainlen=(exp-up).days
        if(domainlen<=365):
            return -1
        else:
            return 1
    except:
        return -1

#Checking the Favicon. Returns 1 if the domain of the favicon image and the URL domain
match else returns -1.
def Favicon(url):
    subDomain, domain, suffix = extract(url)
    b=domain
    try:
        icons = favicon.get(url)
        icon = icons[0]
        subDomain, domain, suffix =extract(icon.url)
        a=domain
        if(a==b):
            return 1
        else:
            return -1
    except:
        return -1

#Checking the Port of the URL. Returns 1 if the port is available else returns -1.
def port(url):
```

```
    try:
        a_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        location=(url[7:],80)
        result_of_check = a_socket.connect_ex(location)
        if result_of_check == 0:
            return 1
        else:
            return -1
        a_socket.close
    except:
        return -1

# HTTPS token in part of domain of URL returns -1, otherwise returns 1
def HTTPS_token(url):
    match=re.search('https://|http://',url)
    if (match.start(0)==0):
        url=url[match.end(0):]
    match=re.search('http|https',url)
    if match:
        return -1
    else:
        return 1


#% of request URL<22% returns 1, otherwise returns -1
def Request_URL(url):
    try:
        subDomain, domain, suffix = extract(url)
        websiteDomain = domain

        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        imgs = soup.findAll('img', src=True)
        total = len(imgs)

        linked_to_same = 0
        avg =0
        for image in imgs:
            subDomain, domain, suffix = extract(image['src'])
            imageDomain = domain
```

```
            if(websiteDomain==imageDomain or imageDomain==''):
                linked_to_same = linked_to_same + 1
        vids = soup.findAll('video', src=True)
        total = total + len(vids)

        for video in vids:
            subDomain, domain, suffix = extract(video['src'])
            vidDomain = domain
            if(websiteDomain==vidDomain or vidDomain==''):
                linked_to_same = linked_to_same + 1
        linked_outside = total-linked_to_same
        if(total!=0):
            avg = linked_outside/total

        if(avg<0.22):
            return 1
        else:
            return -1
    except:
        return -1

#:% of URL of anchor<31% returns 1, % of URL of anchor ≥ 31% and ≤ 67% returns 0,
otherwise returns -1
def URL_of_Anchor(url):
    try:
        subDomain, domain, suffix = extract(url)
        websiteDomain = domain

        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        anchors = soup.findAll('a', href=True)
        total = len(anchors)
        linked_to_same = 0
        avg = 0
        for anchor in anchors:
            subDomain, domain, suffix = extract(anchor['href'])
            anchorDomain = domain
            if(websiteDomain==anchorDomain or anchorDomain==''):
                linked_to_same = linked_to_same + 1
        linked_outside = total-linked_to_same
```

```
        if(total!=0):
            avg = linked_outside/total

        if(avg<0.31):
            return 1
        elif(0.31<=avg<=0.67):
            return 0
        else:
            return -1
    except:
        return 0

#:% of links in <meta>, <script>and<link>tags < 25% returns 1, % of links in <meta>,
#<script> and <link> tags ≥ 25% and ≤ 81% returns 0, otherwise returns -1

def Links_in_tags(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')

        no_of_meta =0
        no_of_link =0
        no_of_script =0
        anchors=0
        avg =0
        for meta in soup.find_all('meta'):
            no_of_meta = no_of_meta+1
        for link in soup.find_all('link'):
            no_of_link = no_of_link +1
        for script in soup.find_all('script'):
            no_of_script = no_of_script+1
        for anchor in soup.find_all('a'):
            anchors = anchors+1
        total = no_of_meta + no_of_link + no_of_script+anchors
        tags = no_of_meta + no_of_link + no_of_script
        if(total!=0):
            avg = tags/total

        if(avg<0.25):
            return -1
```

```
        elif(0.25<=avg<=0.81):
            return 0
        else:
            return 1
    except:
        return 0

#Server Form Handling
#SFH is "about: blank" or empty → phishing, SFH refers to a different domain → suspicious, otherwise → legitimate
def SFH(url):
        try:
            if len('form', action=True)==0:
                return 1
            else :
                for form in ('form', action=True):
                    if form['action'] == "" or form['action'] == "about:blank":
                        return -1
                    elif self.url not in form['action'] and  not in form['action']:
                        return 0
                    else:
                        return 1
        except:
            return -1



#:using "mail()" or "mailto:" returning -1, otherwise returns 1
def Submitting_to_email(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        if(soup.find('mailto:','mail():')):
            return -1
        else:
            return 1
    except:
        return -1

#Host name is not in URL returns -1, otherwise returns 1
def Abnormal_URL(url):
```

```
    subDomain, domain, suffix = extract(url)
    try:
        domain = whois.whois(url)
        hostname=domain.domain_name[0].lower()
        match=re.search(hostname,url)
        if match:
            return 1
        else:
            return -1
    except:
        return -1

#number of redirect page ≤ 1 returns 1, otherwise returns 0
def Redirect(url):
    try:
        request = requests.get(url)
        a=request.history
        if(len(a)<=1):
            return 1
        else:
            return 0

    except:
        return 0



#onMouseOver changes status bar returns -1, otherwise returns 1
def on_mouseover(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')

        no_of_script =0
        for meta in soup.find_all(onmouseover=True):
            no_of_script = no_of_script+1
        if(no_of_script==0):
            return 1
        else:
            return -1
    except:
```

```
        return -1

#right click disabled returns -1, otherwise returns 1
def RightClick(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        if(soup.find_all('script',mousedown=True)):
            return -1
        else:
            return 1
    except:
        return -1

#popup window contains text field → phishing, otherwise → legitimate
def popUpWidnow(url):
        try:
            if re.findall(r"alert\("):
                return 1
            else:
                return -1
        except:
             return -1



#using iframe returns -1, otherwise returns 1
def Iframe(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        nmeta=0
        for meta in soup.findAll('iframe',src=True):
            nmeta= nmeta+1
        if(nmeta!=0):
            return -1
        else:
            return 1
    except:
        return -1
```

```
#:age of domain ≥ 6 months returns 1, otherwise returns -1
def age_of_domain(url):
    try:
        w = whois.whois(url).creation_date[0].year
        if(w<=2018):
            return 1
        else:
            return -1
    except Exception as e:
        return -1

#no DNS record for domain returns -1, otherwise returns 1
def DNSRecord(url):

    subDomain, domain, suffix = extract(url)
    try:
        dns = 0
        domain_name = whois.whois(url)
    except:
        dns = 1

    if(dns == 1):
        return -1
    else:
        return 1

#website rank < 100.000 returns 1, website rank > 100.000 returns 0, otherwise returns -1
def web_traffic(url):
    try:
        rank =
BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url=" +
url).read(), "xml").find("REACH")['RANK']
    except TypeError:
        return -1
    rank= int(rank)
    if (rank<100000):
        return 1
    else:
        return 0
```

```
#:PageRank < 0,2 → phishing, otherwise → legitimate
def Page_Rank(url):
        try:
            prank_checker_response = requests.post("https://www.checkpagerank.net/index.php")

            global_rank = int(re.findall(r"Global Rank: ([0-9]+)",
prank_checker_response.text)[0])
            if global_rank > 0 and global_rank < 100000:
                return 1
            return -1
        except:
            return -1


#webpage indexed by Google returns 1, otherwise returns -1
def Google_Index(url):
    try:
        subDomain, domain, suffix = extract(url)
        a=domain + '.' + suffix
        query = url
        for j in search(query, tld="co.in", num=5, stop=5, pause=2):
            subDomain, domain, suffix = extract(j)
            b=domain + '.' + suffix
        if(a==b):
            return 1
        else:
            return -1
    except:
        return -1


#:number of links pointing to webpage = 0 returns 1, number of links pointing to webpage> 0
#and ≤ 2 returns 0, otherwise returns -1

def Links_pointing_to_page (url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        count = 0
```

```
        for link in soup.find_all('a'):
            count += 1
        if(count>=2):
            return 1
        else:
            return 0
    except:
        return -1

#:host in top 10 phishing IPs or domains returns -1, otherwise returns 1
def Statistical_report (url):
    hostname = url
    h = [(x.start(0), x.end(0)) for x in
regex.finditer('https://|http://|www.|https://www.|http://www.', hostname)]
    z = int(len(h))
    if z != 0:
        y = h[0][1]
        hostname = hostname[y:]
        h = [(x.start(0), x.end(0)) for x in regex.finditer('/', hostname)]
        z = int(len(h))
        if z != 0:
            hostname = hostname[:h[0][0]]

url_match=regex.search('at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|swedd
y\.com|myjino\.ru|96\.lt|ow\.ly',url)
    try:
        ip_address = socket.gethostbyname(hostname)

ip_match=regex.search('146\.112\.61\.108|213\.174\.157\.151|121\.50\.168\.88|192\.185\.217\
.116|78\.46\.211\.158|181\.174\.165\.13|46\.242\.145\.103|121\.50\.168\.40|83\.125\.22\.219|4
6\.242\.145\.98|107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.1
51\.148\.108|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|1
41\.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\.61|213\.19\.128
\.77|62\.113\.226\.131|208\.100\.26\.234|195\.16\.127\.102|195\.16\.127\.157|34\.196\.13\.28|1
03\.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)
```

```
    except:
        return -1

    if url_match:
        return -1
    else:
        return 1

#returning scrapped data to calling function in app.py
def main(url):



    check = [[having_IPhaving_IP_Address
(url),URLURL_Length(url),Shortining_Service(url),having_At_Symbol(url),

double_slash_redirecting(url),Prefix_Suffix(url),having_Sub_Domain(url),SSLfinal_State(url
),

Domain_registeration_length(url),Favicon(url),port(url),HTTPS_token(url),Request_URL(url
),

URL_of_Anchor(url),Links_in_tags(url),SFH(url),Submitting_to_email(url),Abnormal_URL
(url),
            Redirect(url),on_mouseover(url),RightClick(url),popUpWidnow(url),Iframe(url),

age_of_domain(url),DNSRecord(url),web_traffic(url),Page_Rank(url),Google_Index(url),
            Links_pointing_to_page(url),Statistical_report(url)]]


    return check
```

phishing_website.ipynb:

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score,confusion_matrix
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import seaborn as sns
from sklearn import metrics
import warnings
import math
```

Reading the dataset

```
data= pd.read_csv(r"C:\Users\Anand\PycharmProjects\ibm\dataset_website.csv")
data.head()
data.info()
data.nunique()
data.isnull().any()
```

Splitting the data

```
A = data.drop(["Result"],axis =1)
B = data["Result"]
x=data.iloc[:,1:31].values
y=data.iloc[:,-1].values
print(x,y)
```

Testing and Training

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
ML_Model = []
accuracy = []
f1_score = []
recall = []
precision = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
  ML_Model.append(model)
  accuracy.append(round(a, 3))
  f1_score.append(round(b, 3))
  recall.append(round(c, 3))
  precision.append(round(d, 3))
```

Logistic regression

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train,y_train)
pred_test_log=model.predict(x_test)
pred_test_log
y_train_model = model.predict(x_train)
y_test_model = model.predict(x_test)

accuracy_train = metrics.accuracy_score(y_train,y_train_model)
accuracy_test = metrics.accuracy_score(y_test,y_test_model)
print("Logistic Regression : Accuracy on training Data: {:.3f}".format(accuracy_train))
print("Logistic Regression : Accuracy on test Data: {:.3f}".format(accuracy_test))
print()

f1_score_train = metrics.f1_score(y_train,y_train_model)
f1_score_test = metrics.f1_score(y_test,y_test_model)
print("Logistic Regression : f1_score on training Data: {:.3f}".format(f1_score_train))
print("Logistic Regression : f1_score on test Data: {:.3f}".format(f1_score_test))
print()

recall_score_train = metrics.recall_score(y_train,y_train_model)
recall_score_test = metrics.recall_score(y_test,y_test_model)
print("Logistic Regression : Recall on training Data: {:.3f}".format(recall_score_train))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test))
print()

precision_score_train = metrics.precision_score(y_train,y_train_model)
precision_score_test = metrics.precision_score(y_test,y_test_model)
print("Logistic Regression : precision on training Data: {:.3f}".format(precision_score_train))
print("Logistic Regression : precision on test Data: {:.3f}".format(precision_score_test))

storeResults('Logistic Regression',accuracy_test,f1_score_test,
        recall_score_train,precision_score_train)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model_rf=RandomForestClassifier()
model_rf.fit(x_train,y_train)
y_train_model = model_rf.predict(x_train)
y_test_model = model_rf.predict(x_test)
accuracy_train = metrics.accuracy_score(y_train,y_train_model)
accuracy_test = metrics.accuracy_score(y_test,y_test_model)
print("Random Forest : Accuracy on training Data: {:.3f}".format(accuracy_train))
print("Random Forest : Accuracy on test Data: {:.3f}".format(accuracy_test))
print()

f1_score_train = metrics.f1_score(y_train,y_train_model)
f1_score_test = metrics.f1_score(y_test,y_test_model)
print("Random Forest : f1_score on training Data: {:.3f}".format(f1_score_train))
print("Random Forest : f1_score on test Data: {:.3f}".format(f1_score_test))
print()

recall_score_train = metrics.recall_score(y_train,y_train_model)
recall_score_test = metrics.recall_score(y_test,y_test_model)
print("Random Forest : Recall on training Data: {:.3f}".format(recall_score_train))
print("Random Forest : Recall on test Data: {:.3f}".format(recall_score_test))
print()

precision_score_train = metrics.precision_score(y_train,y_train_model)
precision_score_test = metrics.precision_score(y_test,y_test_model)
print("Random Forest: precision on training Data: {:.3f}".format(precision_score_train))
print("Random Forest : precision on test Data: {:.3f}".format(precision_score_test))
storeResults('Random Forest',accuracy_test,f1_score_test,
        recall_score_train,precision_score_train)
```

K - Nearest Neighbour

```
from sklearn.neighbors import KNeighborsClassifier
model= KNeighborsClassifier(n_neighbors=3)
```

```
model.fit(x_train,y_train)
pred_test_knn=model.predict(x_test)
pred_test_knn
y_train_model = model.predict(x_train)
y_test_model = model.predict(x_test)
accuracy_train = metrics.accuracy_score(y_train,y_train_model)
accuracy_test = metrics.accuracy_score(y_test,y_test_model)
print("KNN : Accuracy on training Data: {:.3f}".format(accuracy_train))
print("KNN : Accuracy on test Data: {:.3f}".format(accuracy_test))
print()

f1_score_train = metrics.f1_score(y_train,y_train_model)
f1_score_test = metrics.f1_score(y_test,y_test_model)
print("KNN : f1_score on training Data: {:.3f}".format(f1_score_train))
print("KNN : f1_score on test Data: {:.3f}".format(f1_score_test))
print()

recall_score_train = metrics.recall_score(y_train,y_train_model)
recall_score_test = metrics.recall_score(y_test,y_test_model)
print("KNN : Recall on training Data: {:.3f}".format(recall_score_train))
print("KNN : Recall on test Data: {:.3f}".format(recall_score_test))
print()

precision_score_train = metrics.precision_score(y_train,y_train_model)
precision_score_test = metrics.precision_score(y_test,y_test_model)
print("KNN : precision on training Data: {:.3f}".format(precision_score_train))
print("KNN : precision on test Data: {:.3f}".format(precision_score_test))
storeResults('KNN',accuracy_test,f1_score_test,
        recall_score_train,precision_score_train)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
model.fit(x_train,y_train)
pred_test_dt=model.predict(x_test)
```

```
pred_test_dt
y_train_model = model.predict(x_train)
y_test_model = model.predict(x_test)
accuracy_train = metrics.accuracy_score(y_train,y_train_model)
accuracy_test = metrics.accuracy_score(y_test,y_test_model)
print("Decision Tree : Accuracy on training Data: {:.3f}".format(accuracy_train))
print("Decision Tree : Accuracy on test Data: {:.3f}".format(accuracy_test))
print()

f1_score_train = metrics.f1_score(y_train,y_train_model)
f1_score_test = metrics.f1_score(y_test,y_test_model)
print("Decision Tree : f1_score on training Data: {:.3f}".format(f1_score_train))
print("Decision Tree : f1_score on test Data: {:.3f}".format(f1_score_test))
print()

recall_score_train = metrics.recall_score(y_train,y_train_model)
recall_score_test = metrics.recall_score(y_test,y_test_model)
print("Decision Tree : Recall on training Data: {:.3f}".format(recall_score_train))
print("Decision Tree : Recall on test Data: {:.3f}".format(recall_score_test))
print()

precision_score_train = metrics.precision_score(y_train,y_train_model)
precision_score_test = metrics.precision_score(y_test,y_test_model)
print("Decision Tree : precision on training Data: {:.3f}".format(precision_score_train))
print("Decision Tree : precision on test Data: {:.3f}".format(precision_score_test))
storeResults('Decision Tree',accuracy_test,f1_score_test,
        recall_score_train,precision_score_train)
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
pred_test_nb=model.predict(x_test)
pred_test_nb
y_train_model = model.predict(x_train)
```

```
y_test_model = model.predict(x_test)
accuracy_train = metrics.accuracy_score(y_train,y_train_model)
accuracy_test = metrics.accuracy_score(y_test,y_test_model)
print("Naive Bayes : Accuracy on training Data: {:.3f}".format(accuracy_train))
print("Naive Bayes : Accuracy on test Data: {:.3f}".format(accuracy_test))
print()

f1_score_train = metrics.f1_score(y_train,y_train_model)
f1_score_test = metrics.f1_score(y_test,y_test_model)
print("Naive Bayes : f1_score on training Data: {:.3f}".format(f1_score_train))
print("Naive Bayes : f1_score on test Data: {:.3f}".format(f1_score_test))
print()

recall_score_train = metrics.recall_score(y_train,y_train_model)
recall_score_test = metrics.recall_score(y_test,y_test_model)
print("Naive Bayes : Recall on training Data: {:.3f}".format(recall_score_train))
print("Naive Bayes : Recall on test Data: {:.3f}".format(recall_score_test))
print()

precision_score_train = metrics.precision_score(y_train,y_train_model)
precision_score_test = metrics.precision_score(y_test,y_test_model)
print("Naive Bayes : precision on training Data: {:.3f}".format(precision_score_train))
print("Naive Bayes : precision on test Data: {:.3f}".format(precision_score_test))
storeResults('Naive Bayes',accuracy_test,f1_score_test,
        recall_score_train,precision_score_train)
```

Comparison

```
result = pd.DataFrame({ 'ML Model' : ML_Model,
                    'Accuracy' : accuracy,
                    'f1_score' : f1_score,
                    'Recall'   : recall,
                    'Precision': precision,
                   })
result
ascending_result=result.sort_values(by=['Accuracy',
```

'Precision'],ascending=False).reset_index(drop=True)

ascending_result

Saving the model

import pickle

pickle.dump(model_rf,open('Phishing_Websites.pkl','wb'))

8.TESTING

8.1 TEST CASES

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Commnets	TC for Automation(Y/N)	BUG ID	Executed By
OPENPAGE_1	Functional	Home Page	Verify user is able to see the landing page when user can type the URL in box		1.Enter the website URL and click go 2.Type the URL 3.Verify whether it is processing or not		Should display the webpage	Working as expected	Pass		No		All team members
OPENPAGE_2	UI	Home Page	Verify the UI element is responsive		1.Enter the website URL and click go 2.Type or copy paste the URL 3.Check whether the button is responsive or not 4.reload and test simultaneously		Should wait for the response and then get acknowledged	Working as expected	Pass		No		All team members
OPENPAGE_3	Functional	Home page	Verify whether the link is legitimate or not		1.Enter the website URL and click go 2.Type or copy paste the URL 3.Check the Website is legitimate or not 4. observe the result		The user should observe whether the website is legitimate or not	Working as expected	Pass		No		All team members
OPENPAGE_4	Functional	Home page	Verify user is able to access the legitimate website or not		1.Enter the website URL and click go 2.Type or copy paste the URL 3.Check the Website is legitimate or not 4. continue if the website is legitimate or be cautious if it is not legitimate		Application should show that safe webpage or unsafe.	Working as expected	Pass		No		All team members
OPENPAGE_5	Functional	Home page	Testing the website with multiple URL.		1.Enter the website URL https://google.com and click go 2. Type or copy paste the URL to test 3.Check the website is legitimate or not 4.Continue if the website is secure or be cautious if it is not secure		User can able to identify the websites whether it is secure or not	Working as expected	Pass		No		All team members

8.2 USER ACCEPTANCE TESTING

Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the [Web Phishing Detection] project at the time of the release to User Acceptance Testing (UAT).

Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved.

Resolution	Severity1	Severity2	Severity3	Severity4	Subtotal
By Design	5	1	1	1	8
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	7	2	4	10	23
Not Reproduced	0	0	0	0	0
Skipped	0	0	1	1	2
Won'tFix	0	3	2	1	6
Totals	15	9	11	14	49

Test Case Analysis

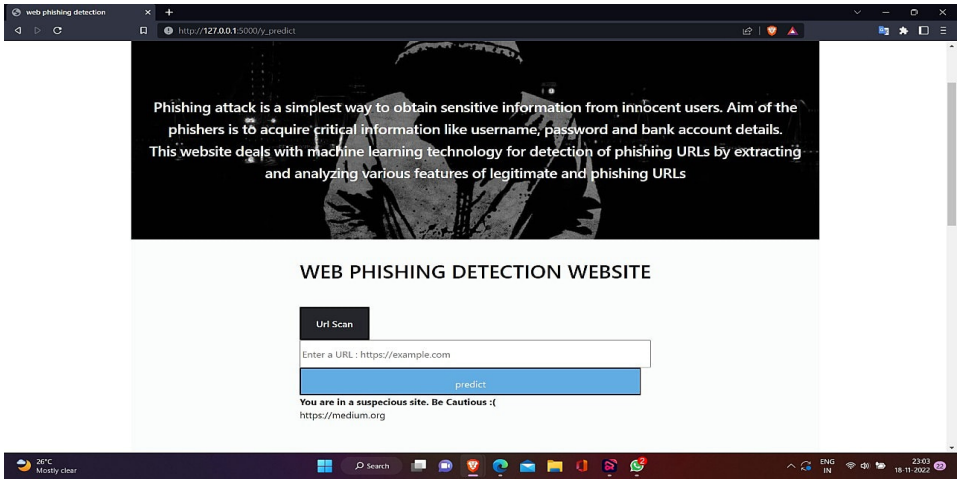
This report shows the number of test cases that have passed, failed, and untested.

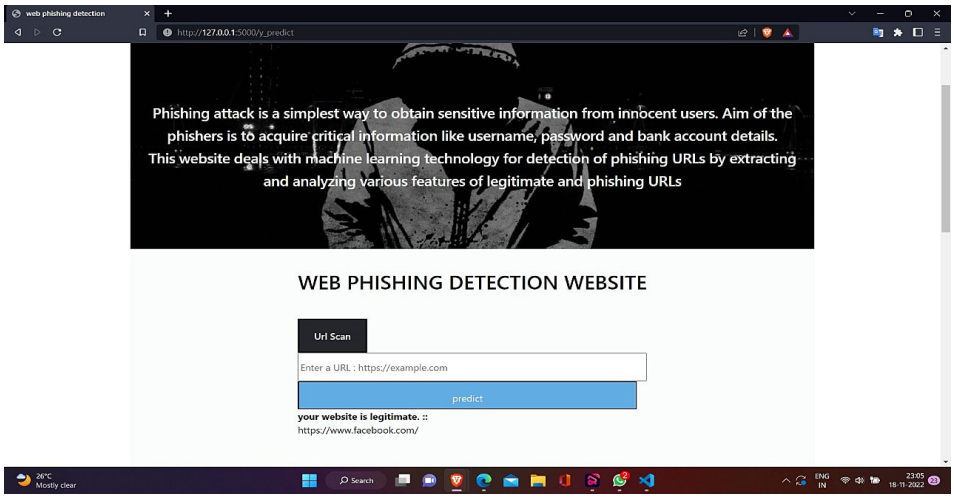
Section	TotalCases	Not Tested	Fail	Pass
ClientApplication	10	0	0	10
Security	10	0	0	10
FinalReportOutput	10	0	0	10
performance	10	0	0	10

9.RESULTS

9.1 PERFORMANCE METRICS

S.No	Parameter	Values
1.	Model summary	As Phishing attacks are increasing these days, the need for detecting phishing website is necessary.The Random forest algorithm is used for building the model. Thus proposed model successfully detects the phishing and legitimate websites using machine learning algorithm.
2.	Accuracy	Training accuracy - 95% - 98% Validation accuracy - 96%





10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

1. Protects the privacy of customers and prevents from loss of data or money.
2. Prevents users from fraudulent websites
3. Handles more number of users simultaneously.
4. Credit and banking details can be prevented by this website.

DISADVANTAGES:

1. The model may fail to identify the phishing websites in case when the phishers use the embedded objects (i.e., JavaScript, images, Flash, etc.) to obscure the textual content and HTML coding from the anti-phishing solutions.
2. The model takes more time to detect the accuracy of the phishing websites.
3. The model not able to detect the website which the phishing websites include malware.

11.CONCLUSION

Phishing website attacks are the most important modern challenge for users, and they continue to show a rising trend in recent years. Blacklist/whitelist techniques are the traditional way to relieve such threats. However, these methods fail to detect non-blacklisted phishing websites (zero-hour attacks). The machine learning techniques are being used to increase detection efficiency and reduce the misclassification ratio. However, some of them extract features from third-party services, search engines, website traffic, etc., which are complicated and difficult to access. In this model,We used features from various domain spanning from URL to HTML tags of the webpage, from embedded URLs to favicon, and databases like google-index, DNS-record, Page-rank, etc. to check the traffic and status of the website. We propose a machine learning-based approach using Random Forest Algorithm can preciously and accuracy of 96% to detect phishing websites of the given webpage.

The model has ability to detect the web3 pages which is more popular and seems to be a perfect target for phishing attacks like phishing scams on the blockchain.

12.FUTURE SCOPE

The project work presented here has created for 11,055 URLs, and 30 features for each URL has been used .Random forest algorithm was used to find the phishing website and 96% of accuracy has been obtained. There are so many features present to check the website. In future more URL's can be added and features are also updated with new algorithm techniques. The accuracy might also be increased.

13.APPENDIX

SOURCE CODE:

Project source code link:https://colab.research.google.com/drive/1sK2RWKhnBvmer-gQhtlQ3Y9O1s_SVi2h

github link:https://github.com/IBM-EPBL/IBM-Project-42635-1660672351

DEMO VIDEO:

Demo Video Link:https://youtu.be/ANxtJjOVHoM