PROJECT REPORT

INTRODUCTION

1.1 PROJECT OVERVIEW

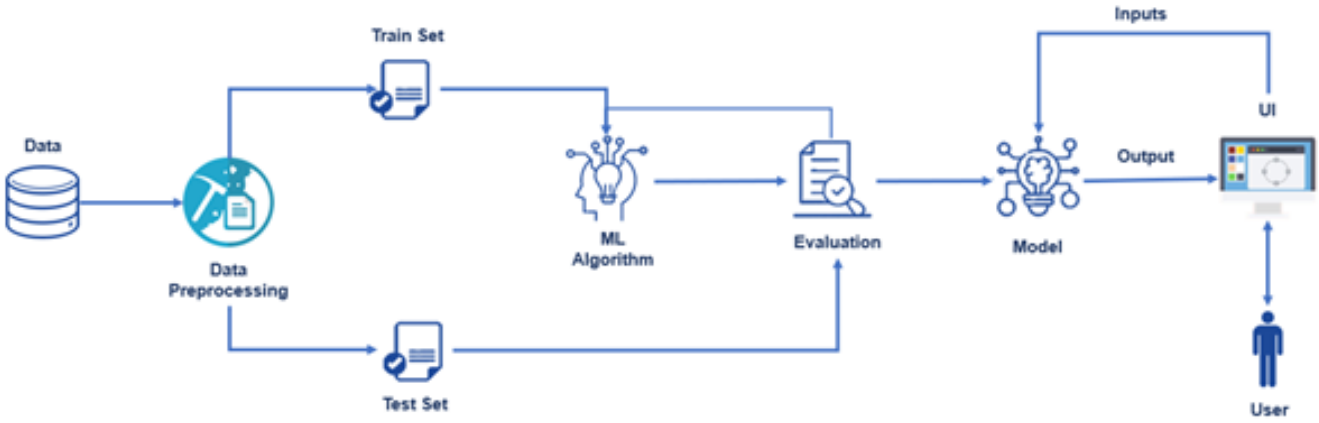
There are a number of users who purchase products online and make payments through e-banking. There are e-banking websites that ask users to provide sensitive data such as username, password & credit card details, etc., often for malicious reasons. This type of e-banking website is known as a phishing website. Web service is one of the key communications software services for the Internet. Web phishing is one of many security threats to web services on the Internet.

Common threats of web phishing:

1. Web phishing aims to steal private information, such as usernames, passwords, and credit card details, by way of impersonating a legitimate entity.
2. It will lead to information disclosure and property damage.
3. Large organizations may get trapped in different kinds of scams.
4. This Guided Project mainly focuses on applying a machine-learning algorithm to detect Phishing websites.

In order to detect and predict e-banking phishing websites, we proposed an intelligent, flexible and effective system that is based on classification algorithms. We implemented classification algorithms and techniques to extract the phishing datasets criteria to classify their legitimacy. The e-banking phishing website can be detected based on some important characteristics like URL and domain identity, security and encryption criteria in the final phishing detection rate. Once a user enters a URL in our website, our system will use a machine learning algorithm to detect whether the entered URL is a phishing URL or not.

TECHNICAL ARCHITECTURE:



1.2 PURPOSE:

URL is the first thing to analyse a website to decide whether it is a phishing or not. As we mentioned before, URLs of phishing domains have some distinctive points. Features which are related to these points are obtained when the URL is processed. Some of URL-Based Features are given below.

1. Digit count in the URL
2. Total length of URL
3. Checking whether the URL is Typosquatted or not. (google.com → goggle.com)
4. Checking whether it includes a legitimate brand name or not (apple-icloud-login.com)
5. Number of subdomains in URL
6. Is Top Level Domain (TLD) one of the commonly used one?

2.LITERATURE SURVEY:

INTRODUCTION:

Phishing attacks are the practice of sending fraudulent communications that appear to come from a reputable source. It is usually done through email. The goal is to steal sensitive data like credit card and login information, or to install malware on the victim's machine. For example, a system can be technically secure enough against password theft, however unaware end users may leak their passwords if an attacker asked them to update their passwords via a given Hypertext Transfer Protocol (HTTP) link, which ultimately threatens the overall security of the system.

Since phishing attacks aim at exploiting weaknesses found it is difficult to mitigate them. On the other hand, software phishing detection techniques are evaluated against bulk phishing attacks, which makes their performance practically unknown with regards to targeted forms of phishing attacks. These limitations in phishing mitigation techniques have practically resulted in security breaches against several organizations including leading information security providers

This survey begins by:

• Defining the phishing problem. It is important to note that the phishing definition in the literature is not consistent, and thus a comparison of a number of definitions is presented.

• Categorizing anti-phishing solutions from the perspective of phishing campaign life[1]cycle. This presents the various anti-phishing solution categories such as detection. It is important to view the overall anti-phishing picture from a high-level perspective before diving into a particular technique, namely: phishing detection techniques (which is the scope of this survey).

• Presenting evaluation metrics that are commonly used in the phishing domain to evaluate the performance of phishing detection techniques. This facilitates the comparison between the various phishing detection techniques.

• Presenting a literature survey of anti-phishing detection techniques, which incorporates software detection techniques as well as user-awareness techniques that enhance the detection process of phishing attacks.

• Presenting a comparison of the various proposed phishing detection techniques in

the literature

HISTORY:

According to APWG, the term phishing was coined in 1996 due to social engineering attacks against America On-line (AOL) accounts by online scammers. The term phishing comes from fishing in a sense that fishers (i.e. attackers) use a bait (i.e. socially-engineered messages) to fish (e.g. steal personal information of victims). However, it should be noted that the theft of personal information is mentioned here as an example, and that attackers are not restricted by that as previously defined in Section II. The origins of the ph replacement of the character f in fishing is due to the fact that one of the earliest forms of hacking was against telephone networks, which was named Phone Phreaking. As a result, ph became a common hacking character replacement of f. According to APWG, stolen accounts via phishing attacks were also used as a currency between hackers by 1997 to trade hacking software in exchange of the stolen accounts. Phishing attacks were historically started by stealing AOL accounts, and over the years moved into attacking more profitable targets, such as on-line banking and e[1]commerce services. Currently, phishing attacks do not only target system end users, but also technical employees at service providers, and may deploy sophisticated techniques such as MITB attacks.

PHISHING MOTIVES:

According to Weider D. et. al. [6], the primary motives behind phishing attacks, from an attacker's perspective, are:

• Financial gain: phishers can use stolen banking credentials to their financial benefits.

• Identity hiding: instead of using stolen identities directly, phishers might sell the identities to others whom might be criminals seeking ways to hide their identities and activities (e.g. purchase of goods).

• Fame and notoriety: phishers might attack victims for the sake of peer recognition

CHALLENGES:

Because the phishing problem takes advantage of human ignorance or naivety with regards to their interaction with electronic communication channels (e.g. E-Mail, HTTP, etc. . .), it is not an easy problem to permanently solve. All of the proposed solutions attempt to minimize the impact of phishing attacks. From a high-level perspective, there are generally two commonly suggested solutions to mitigate phishing attacks

• User education; the human is educated in an attempt to enhance his/her classification accuracy to correctly identify phishing messages, and then apply proper actions on the correctly classified phishing messages, such as reporting attacks to system administrators

• Software enhancement; the software is improved to better classify phishing messages on behalf of the human, or provide information in a more obvious way so that the human would have less chance to ignore it

The challenges with both of the approaches are:

• Non-technical people resist learning, and if they learn they do not retain their knowledge permanently, and thus training should be made continuous. Some software solutions, such as authentication and security warnings, are still dependent on user behaviour. If users ignore security warnings, the solution can be rendered useless.

• Phishing is a semantic attack that uses electronic communication channels to deliver content with natural languages (e.g. Arabic, English, French, etc. . .) to persuade victims to perform certain actions. The challenge here is that computers have extreme difficulty in accurately understanding the semantics of natural languages.

DETECTION APPROACHES:

In this survey, we consider any anti-phishing solution that aims to identify or classify phishing attacks as detection solutions. This includes:

• User training approaches — end-users can be educated to better understand the nature of phishing attacks, which ultimately leads them into correctly identifying phishing and non-phishing messages. This is contrary to the categorization where

user training was considered a preventative approach. However, user training approaches aim at enhancing the ability of end-users to detect phishing attacks, and thus we categorize them under “detection”.

• Software classification approaches — these mitigation approaches aim at classifying phishing and legitimate messages on behalf of the user in an attempt to bridge the gap that is left due to the human error or ignorance. This is an important gap to bridge as user-training is more expensive than automated software classifiers, and user training may not be feasible in some scenarios (such as when the user base is huge, e.g. PayPal, eBay, etc. . .)

DETECTION OF PHISHING ATTACKS:

Inputs to the decision making process are:

• External information: could be anything learned through the User Interface (UI) (Web/mail client and their content), or expert advice. The phisher only has control over what is presented by the UI. Usually, the user does not ask for expert advice unless he is in doubt (i.e. if a user is convinced that a phishing site is legitimate, he might not ask for expert advice in the first place).

• Knowledge and context: the user’s current understanding of the world, which is built over time (e.g. news, past experience).

• Expectation: users have expectations based on their understanding and the outcome of their actions. During the decision making process, two types of decisions can be made, which are: • Planning a series of actions to be taken. • Deciding on the next action in sequence to be taken. Thisis influenced by the outcome resulting from the previous action.

Each of the two types of decisions mentioned above, follow the following steps:

• Construction of perception: constructed through the context where the user reads (say) an email message. Such as, senders/recipients, conversation cause, or suggested actions by the email. In legitimate messages, there are no inconsistencies between the reality and message claims (e.g. senders are the real senders whom they claim to be, and suggested actions by email content does what it says). However, in phishing messages there are inconsistencies(e.g. if the sender’s ID is spoofed, or the message’s content claims to fix a problem while attempting, in

reality, to obtain personal information). If the end-user discovers inconsistencies in a given phishing message, the phishing attack would then fail to persuade the end user.

• Generation of possible solutions: users usually find solutions through available resources. However, with phishing emails, the user is not requested to generate a possible solution in the first place, as the phisher already suggests a solution to the user. For example, if the phishing email content presents a problem, such as account expiry, it will also present a solution, such as activating the account through logging in a URL from which expiry is prevented.

• Generation of assessment criteria: different users have different criteria that reflects how they view the world, their emotional state, personal preferences, etc. . . . As the paper claims, most phishing attempts do not take into account such details, but rely on generic common-sense criteria instead; for example: an attacker might place a tick box labelled “Secure login” to meet a security criterion most users require. Phishing attacks aim to match user criteria as much as possible

PHISHING DETECTION BY BLACKLISTS:

Blacklists are frequently updated lists of previously detected phishing URLs, Internet Protocol (IP) addresses or keywords. Whitelists, on the other hand, are the opposite, and could be used to reduce FP rates. Blacklists do not provide protection against zero-hour phishing attacks as a site needs to be previously detected first in order to be blacklisted. However, blacklists generally have lower FP rates than heuristics

PROPOSED RULES FOR DETECTION:

The proposed rules fall under:

• Analysis performed on URL that fall within the email’s body.

• Analysis performed on email headers. The proposed rules are (where positive indicates phishiness):

• Rule 1: If a URL is a login page that is not a business’s real login page, the result is positive. The paper specifies that this is analyzed based on data returned from search engines.

• Rule 2: If the email is formatted as HTML, and an included URL uses Transport Layer Security (TLS) while the actual Hypertext Reference (HREF) attribute does not use TLS, then the result is positive.

• Rule 3: If the host-name portion of a URL is an IP address, the result is positive.

• Rule 4: If a URL mentions an organization's name (e.g. PayPal) in a URL path but not in the domain name, the result is positive.

• Rule 5: If URL's displayed domain does not match the domain name as specified in HREF attribute, the result is positive.

• Rule 6: If the received SMTP header does not include the organization's domain name, the result is positive.

• Rule 7: If inconsistencies are found in a non-image URL's domain portion, the result is positive.

• Rule 8: If inconsistencies are found in Whois records of non-image URL's domain portion, the result is positive.

• Rule 9: If inconsistencies are found in image URL's domain portion, the result is positive.

• Rule 10: If inconsistencies are found in Whois records of image URL's domain portion, the result is positive.

• Rule 11: If the page is not accessible, the result is positive

CONCLUSION:

User education or training is an attempt to increase the technical awareness level of users to reduce their susceptibility to phishing attacks. However, the human factor is broad and education alone may not guarantee a positive behavioural response. This survey reviewed a number of anti-phishing software techniques. Some of the important aspects in measuring phishing solutions are:

• Detection accuracy with regards to zero-hour phishing attacks. This is due to the fact that phishing websites are mostly short-lived and detection at hour zero is critical.

• Low false positives. A system with high false positives might cause more harm

than good. Moreover, end-users will get into the habit of ignoring security warnings if the classifier is often mistaken.

Generally, software detection solutions are:

• Blacklists.

• Rule-based heuristics.

• Visual similarity.

• Machine Learning-based classifiers.

The Machine Learning-based detection techniques achieved high classification accuracy for analyzing similar data parts to those of rule-based heuristic techniques

2.2 REFERNCES:

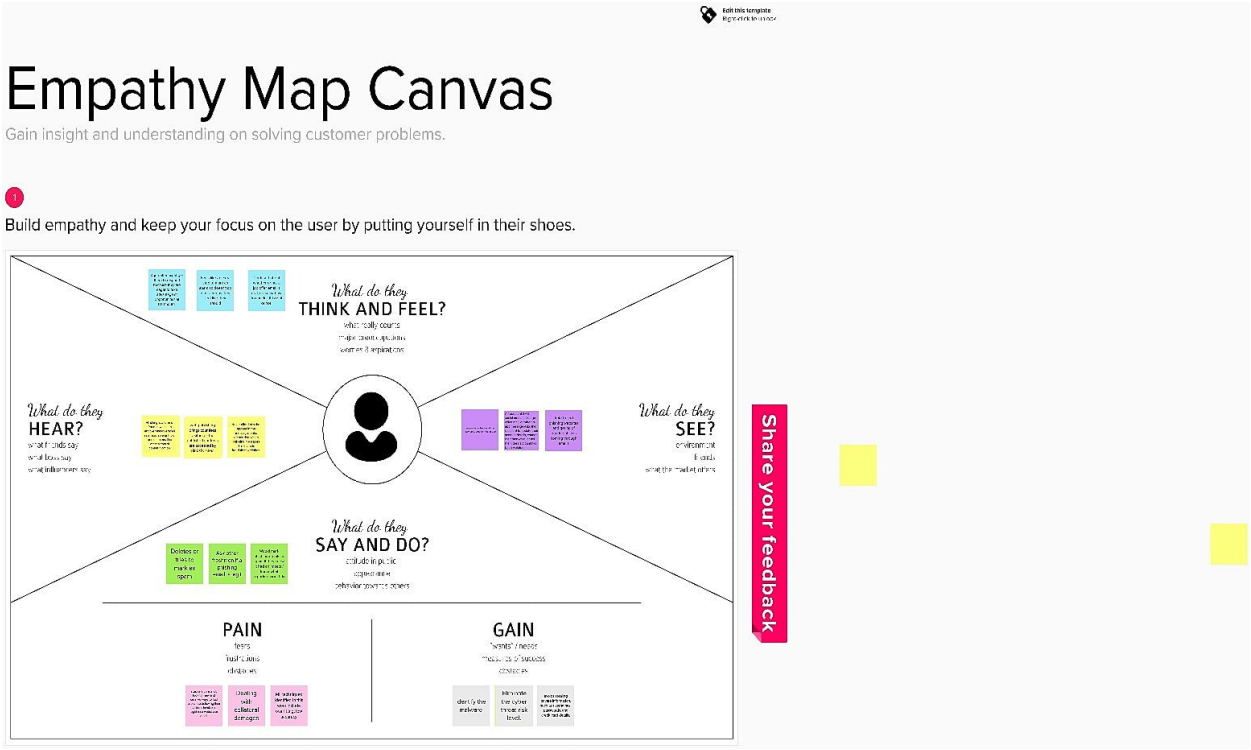
[1] S. Sheng, M. Holbrook, P. Kumaraguru, L. F. Cranor, and J. Downs, “Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions,” in Proceedings of the 28th international conference on Human factors in computing systems, ser. CHI ’10. New York, NY, USA: ACM, 2010, pp. 373–382.

[2] B. Krebs, “HBGary Federal hacked by Anonymous,” http: //krebsonsecurity.com/2011/02/hbgary-federal-hacked-by-anonymous/, 2011, accessed December 2011.

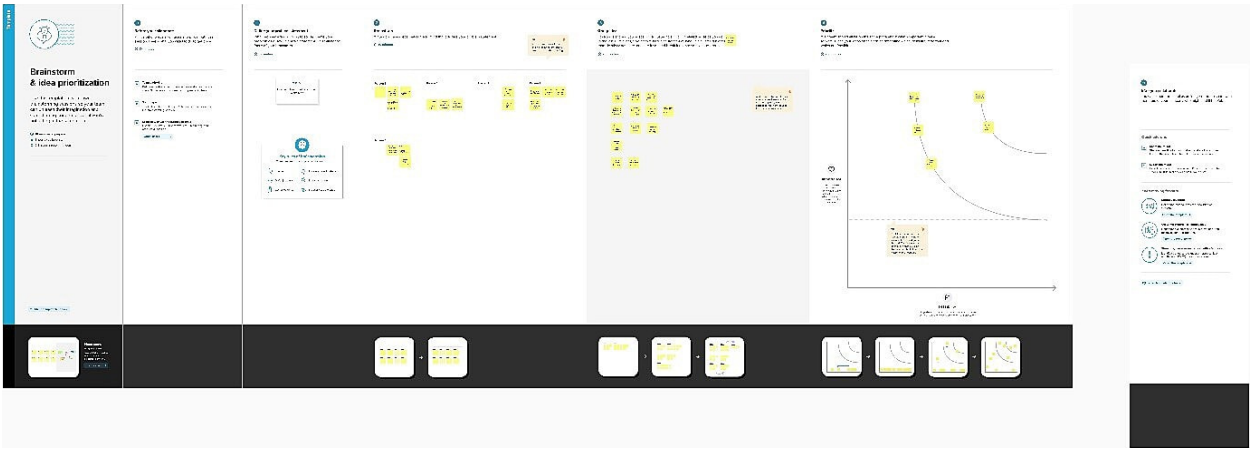
2.3 PROBLEM STATEMENT DEFINITION:

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 IDEATION & BRAINSTORMING



3.3 PROPOSED SOLUTION

PROBLEM STATEMENT:

Internet has dominated the world by dragging half of the world's population exponentially into the cyber world. With the booming of internet transactions, cybercrimes rapidly increased and with anonymity presented by the internet, Hackers attempt to trap the end-users through various forms such as phishing, SQL injection, malware, man-in-the-middle, domain name system tunnelling, ransomware, web trojan, and so on. Among all these attacks, phishing reports to be the most deceiving attack. Our main aim of this paper is classification of a phishing website with the aid of various machine learning techniques to achieve maximum accuracy and concise model.

IDEA/SOLUTION DESCRIPTION:

There are two approaches that are typically used in detecting phishing websites. The first approach is

typically based on a blacklist, where in the given URL is compared with the URLs present in the
blacklist. The other part of this approach is that the blacklist usually cannot identify all phishing sites,
hence a new fraudulent website is launched. The alternate or the second approach is referred to as
heuristic based methods, where few of the features are collected from the sites to distinguish it as
either phishing or legitimate.

NOVELTY/UNIQUENESS:

In terms of accuracy, it was primarily due to the capability of the proposed PSO based feature
weighting to successfully weight the website features used for enhancing phishing website detection. In
addition to the classification accuracy, we can have the TPR, TNR, FPR, FNR of machine learning
classifiers before and after applying the proposed PSO based feature weighting.

SOCIAL IMPACT/CUSTOMER SATISFACTION:

An exhaustive systematic search was performed on all the indexing databases. The state-of-the-art
research related to the web phishing detections was collected. The papers were classified based on the
methodologies. A taxonomy was derived by performing a deep scan on the classified papers. The
contributions listed in this survey are exhaustive and lists all the state-of-the-art development in this
area.

BUSINESS MODEL (FINANCIAL BENEFIT):

Phishing attacks are categorized according to Phisher’s mechanism for trapping alleged users. Several
forms of these attacks are keyloggers, DNS toxicity, Etc., [2]. The initiation processes in social

engineering include online blogs, short message services (SMS), social media platforms that use web
2.0 services, such as Facebook and Twitter, file-sharing services for peers, Voice over IP (VoIP)
systems where the attackers use caller spoofing IDs [3, 4]. Each form of phishing has a little
difference in how the process is carried out in order to defraud the unsuspecting consumer. E-mail
phishing attacks occur when an attacker sends an e-mail with a link to potential users to direct them to
phishing websites.

SCALABILITY OF SOLUTION:

The methods are evaluated in terms of learning rate, accuracy, and precision. It presents the learning
rate of the methods during the training phase. The performance of three detectors during the training
phase are similar. It is evident that the learning ability of methods are same. Authors maintained similar parameters for all detectors. The learning rate of LURL is reasonable comparing to other two
methods. It indicates that ML based methods able to scan an average of 84% of dataset to learn the
environment at the rate of 1.0.

3.4 PROPOSED SOLUTION FIT:

1.CUSTOMER SEGMENTS ➢ It allows you to fine tune your message. ➢ Increase your revenue. ➢ You can increase the awareness for your brand.	**6.CUSTOMER CONSTRAINTS** ➢ Don't click on that link. ➢ Customers should know what a phising scam looks like. ➢ Don't give information to an unsecured site. ➢ Rotate passwords regularly inorder to avoid phishing.	**5.AVAILABLE SOLUTIONS** ➢ Use anti-phishing protection and anti-spam software to protect yourself when malicious messages slip through your computer. Anti-malware is included to prevent other types of threats.
2.JOBS TO BE DONE/PROBLEMS ➢ The main purpose of this research is to secure a people from hacking or to secure their data or information from the unauthorized person.	**9.PROBLEM ROOT CAUSE** ➢ Users lacks security awareness. ➢ Criminals are following the money. ➢ You're not performing sufficient due diligence.	**7.BEHAVIOUR** ➢ The emails makes unrealistic threats or demands. ➢ A mismatched or dodgy URL. ➢ There's a catch. ➢ Poor spelling and grammer.
3.TRIGGERS ➢ The problem with phishing is that attackers constantly look for new and creative ways to fool users into believing their actions involve a legitimate website or email.	**10.YOUR SOLUTION** There is a build in policy and definition for impersonation but we also created a single policy/definition with our c-level users information. Mimecast is a full email security platform so it offers a large array of security setups from email, weblinks, etc.	**8.CHANNELS OF BEHAVIOUR** ➢ The customers are asked for a sensitive information and rushing us to do it as soon as possible for forging.

4.EMOTIONS:BEFORE/AFTER		8.2 OFFLINE
➢ Secure the customers. ➢ Save the information made them feel safe.		➢ Don't be tempted by those pop-ups. ➢ Have a data security platform To spot signs of an attacks.

4.REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIn
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login to the portal
FR-4	Search for phishing data of the user	Basic credentials of the user
FR-5	Interface response	Prediction of data leakage
FR-6	Security measures	Recover the data that was leaked

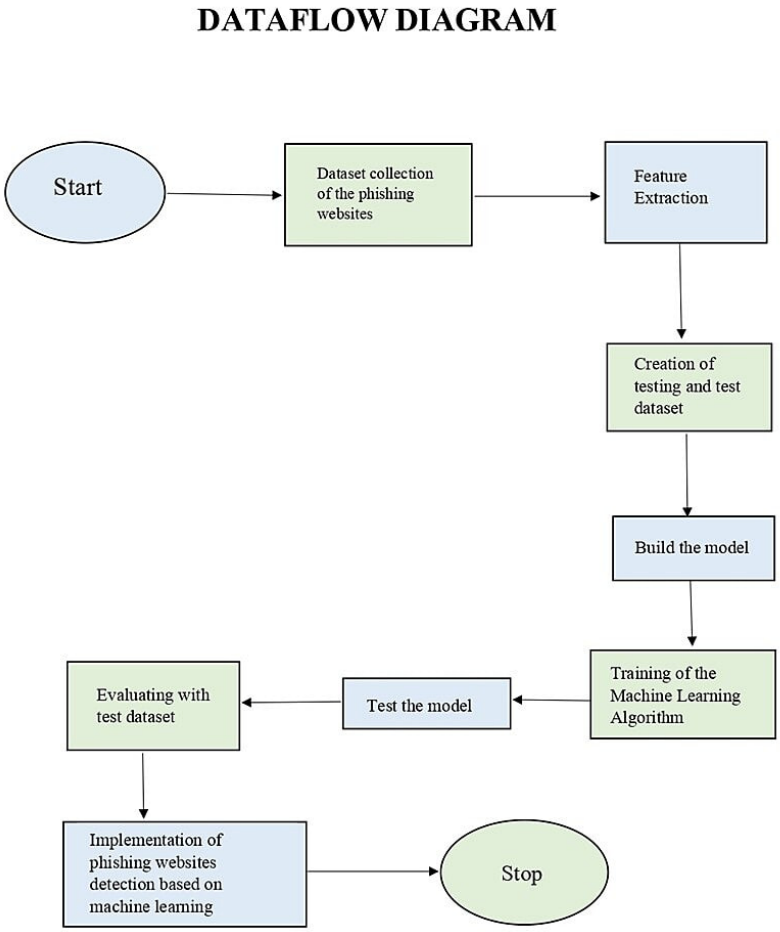
4.2 NON FUNCTIONAL REQUIREMENTS

Following are the non-functional requirements of the proposed solution.

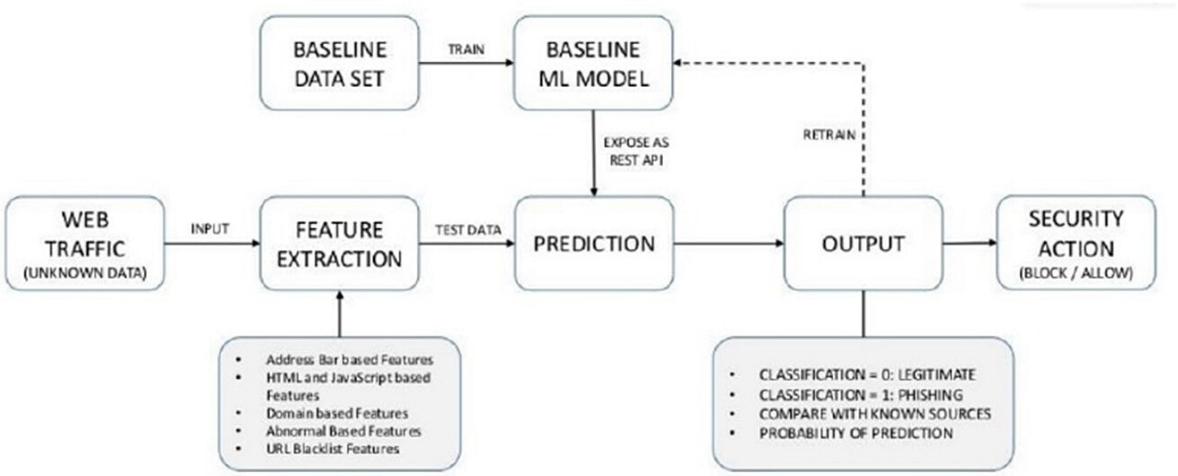
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	It can be used for different attributes.
NFR-2	**Security**	Security questions will need to be setup when a user registers as a user. Security questions will need to be answered when a user forgets their password. The system will have a number of unique users and each user has access constraints
NFR-3	**Reliability**	Phishing websites can be detected with 97.95% accuracy using a Light GBM classifier and the complete set of the 54 features selected, when it was evaluated on PILWD dataset.
NFR-4	**Performance**	The system should be fast and accurate. The system will handle expected and non-expected errors in a manner that will prevent information loss and long downtime periods.
NFR-5	**Availability**	It is available to all end users
NFR-6	**Scalability**	It can hold requests of more than 500 at a time.

5.PROJECT DESIGN

5.1 DATA FLOW DIAGRAM



5.2 SOLUTION & TECHNICAL ARCHITECTURE



5.3 USER STORIES

CUSTOMER JOURNEY MAP

Product Name: **Web Phishing Detection**

PHASES	Dataset preparation and feature extraction	Training&validation of individual algorithm	Ensemble design and comparative study
ACTIVITY	Poor grammer and misspelled words in an email can be red flags.	Encourage your clients to look for ant unusual or odd requests in their emails.	It is used to detect inconsistencies in email addresses,links and domain names
TOUCHPOINT	Data gathering stage is done manually by using google crawler& phishtank each of data gathering methods were tested to ensure valid output	Training and validation of individual algorithm is used to test the performance of individual classifiers in varying dataset.	Ensemble design have two parts which is ensemble design & comparative study between best ensemble and best individual classifier.
EXPERIENCE	Percentage of devices that are appropriately patched.	We have greatly sensitized our users to the potential for security incidents to exist whether they actually exist or not.	Limitations in the state of art detection approaches are identified and presented leading to future research directions.

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

Finished tasks (Milestones & Activities)

Milestones	Activities	Description
Ideation Phase	Literature Survey	Literature survey on the selected project & information gathering
	Empathy Map	Prepare Empathy map to capture the user Panis & Gains, prepare list of problem statement
	Ideation	Organizing the brainstroming session and prioritise the top 3 ideas based on feasibility & Importance
Project Design Phase I	Proposed Solution	Prepare proposed solution document which includes novelty, feasibility of ideas, business model, social impact, Scalability of solution
	Problem Solution Fit	Prepare problem solution fit document
	Solution Architecture	Prepare solution architecture document
Project Design Phase II	Customer Journey	Prepare customer journey map to understand the user interactions & experience with the application
	Functional requirement	Prepare functional & non functional requirement document
	Data Flow Diagram	Prepare Data Flow Diagram and user stories
	Technology architecture	Draw the technology architecture diagram
Project Planning Phase	Milestones & Activity list	Prepare milestones and activity list of the project
	Sprint Delivery Plan	Prepare sprint delivery plan

Remaining tasks (Milestones & Activities) to be completed

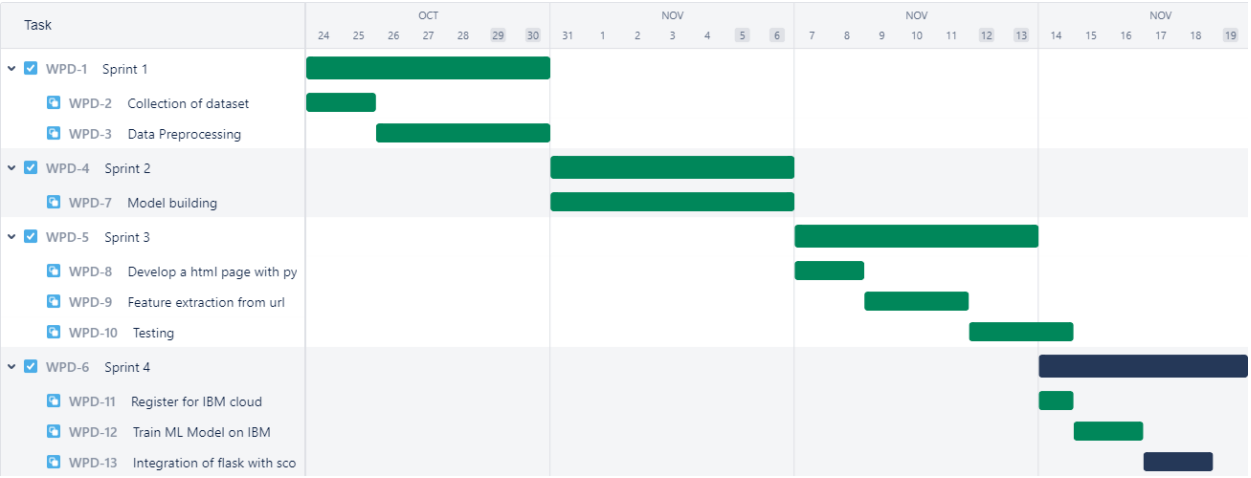
Milestones	Activities	Description
Project Development Phase	Delivery of Sprint – 1,2,3,4	To develop the code and submit the developed code by testing it
Setting up App environment	Create IBM Cloud account	Signup for an IBM Cloud account
	Create flask project	Getting started with Flask to create project
	Install IBM Cloud CLI	Install IBM Command Line Interface
	Docker CLI Installation	Installing Docker CLI on laptop
	Create an account in sendgrid	Create an account in sendgrid. Use the service as email integration to our application for sending emails
Implementing web Application	Create UI to interact with Application	Create UI • Registration page • Login page • URL text box • Displaying output
	Create IBM DB2 & connect with python	Create IBM DB2 service in IBM Cloud and connect with python code with DB
Deployment of App in IBM Cloud	Check the URL	The URL which needs to check should be entered on the column provided
	Compare using ML	The Processed dataset checks the given URL is phishing or not
	Output	By Comparing the given URL with the already defined one, the machine will produce the output as phishing website or not

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Collection of Dataset	USN-1	We have to select or identify a dataset which contains a set of features through which a phishing website can be identified.	2	High	2
Sprint-1		USN-2	We have to download the dataset	1	High	1
Sprint-1	Data Preprocessing	USN-3	In this we will be pre-processing the dataset that is collected.	1	High	1
Sprint-1		USN-4	This includes Handling the null values, Handling the categorical values if any, Normalize the data if required. Identifying the dependent and independent variables, Split the dataset into train and test sets.	3	High	3

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint -2	Model Building	USN-5	Here we have to choose the appropriate algorithm or model .The dataset which we are using is a Classification dataset ,So we are using the following algorithms. Logistic Regression, Random Forest Regression / Classification, Decision Tree Regression Classification, K-Nearest Neighbors, Support Vector Machine	5	High	5
Sprint-3		USN-6	In order to get appropriate predictions, the dataset can be trained with any of the above algorithms.	5	Medium	5
Sprint-3	Application Building	USN-7	Here we have to Building an Application to integrate the model	2	Low	2
Sprint-4		USN-8	Here we will be integrating it to a web application so that normal users can also use it to know if any website is phishing or safe	3	High	3
Sprint-1		USN-9	the user provides any website URL to check and the corresponding parameter values are generated by analysing the URL using which legitimate websites are detected.	2	High	2
Sprint-2	Train the model	USN-10	In this we have to build a Machine Learning Model and deploy it on the IBM Cloud.	5	High	5

6.3 REPORTS FROM JIRA



7.CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 FEATURE

490 lines (432 sloc) | 16.3 KB

```
1   import ipaddress
2   import re
3   import urllib.request
4   from bs4 import BeautifulSoup
5   import socket
6   import requests
7   from googlesearch import search
8   import whois
9   from datetime import date, datetime
10  import time
11  from dateutil.parser import parse as date_parse
12  from urllib.parse import urlparse
13
14  class FeatureExtraction:
15      features = []
16      def __init__(self,url):
17          self.features = []
18          self.url = url
19          self.domain = ""
20          self.whois_response = ""
21          self.urlparse = ""
22          self.response = ""
23          self.soup = ""
24
25          try:
26              self.response = requests.get(url)
27              self.soup = BeautifulSoup(self.response.text, 'html.parser')
```

```
        except:
            pass

        try:
            self.urlparse = urlparse(url)
            self.domain = self.urlparse.netloc
        except:
            pass

        try:
            self.whois_response = whois.whois(self.domain)
        except:
            pass




        self.features.append(self.UsingIp())
        self.features.append(self.longUrl())
        self.features.append(self.shortUrl())
        self.features.append(self.symbol())
        self.features.append(self.redirecting())
        self.features.append(self.prefixSuffix())
        self.features.append(self.SubDomains())
        self.features.append(self.Hppts())
        self.features.append(self.DomainRegLen())
        self.features.append(self.Favicon())


        self.features.append(self.NonStdPort())
        self.features.append(self.HTTPSDomainURL())
        self.features.append(self.RequestURL())
        self.features.append(self.AnchorURL())
        self.features.append(self.LinksInScriptTags())
        self.features.append(self.ServerFormHandler())
        self.features.append(self.InfoEmail())
        self.features.append(self.AbnormalURL())
        self.features.append(self.WebsiteForwarding())
        self.features.append(self.StatusBarCust())
```

```
67
68          self.features.append(self.DisableRightClick())
69          self.features.append(self.UsingPopupWindow())
70          self.features.append(self.IframeRedirection())
71          self.features.append(self.AgeofDomain())
72          self.features.append(self.DNSRecording())
73          self.features.append(self.WebsiteTraffic())
74          self.features.append(self.PageRank())
75          self.features.append(self.GoogleIndex())
76          self.features.append(self.LinksPointingToPage())
77          self.features.append(self.StatsReport())
78
79
80      # 1.UsingIp
81      def UsingIp(self):
82          try:
83              ipaddress.ip_address(self.url)
84              return -1
85          except:
86              return 1
87
88      # 2.longUrl
89      def longUrl(self):
90          if len(self.url) < 54:
91              return 1
92          if len(self.url) >= 54 and len(self.url) <= 75:
93              return 0
94          return -1
95
96      # 3.shortUrl
97      def shortUrl(self):
98          match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'
99                      'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'
100                     'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'
101                     'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
102                     'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
103                     'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'
104                     'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|tr\.im|link\.zip\.net', self.url)
```

```
105             if match:
106                 return -1
107             return 1
108
109         # 4.Symbol@
110         def symbol(self):
111             if re.findall("@",self.url):
112                 return -1
113             return 1
114
115         # 5.Redirecting//
116         def redirecting(self):
117             if self.url.rfind('//')>6:
118                 return -1
119             return 1
120
121         # 6.prefixSuffix
122         def prefixSuffix(self):
123             try:
124                 match = re.findall('\-', self.domain)
125                 if match:
126                     return -1
127                 return 1
128             except:
129                 return -1
130
131         # 7.SubDomains
132         def SubDomains(self):
133             dot_count = len(re.findall("\.", self.url))
134             if dot_count == 1:
135                 return 1
136             elif dot_count == 2:
137                 return 0
138             return -1
139
140         # 8.HTTPS
141         def Hppts(self):
142             try:
143                 https = self.urlparse.scheme
```

```
            if 'https' in https:
                return 1
            return -1
        except:
            return 1

    # 9.DomainRegLen
    def DomainRegLen(self):
        try:
            expiration_date = self.whois_response.expiration_date
            creation_date = self.whois_response.creation_date
            try:
                if(len(expiration_date)):
                    expiration_date = expiration_date[0]
            except:
                pass
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            age = (expiration_date.year-creation_date.year)*12+ (expiration_date.month-creation_date.month)
            if age >=12:
                return 1
            return -1
        except:
            return -1

    # 10. Favicon
    def Favicon(self):
        try:
            for head in self.soup.find_all('head'):
                for head.link in self.soup.find_all('link', href=True):
                    dots = [x.start(0) for x in re.finditer('\.', head.link['href'])]
                    if self.url in head.link['href'] or len(dots) == 1 or self.domain in head.link['href']:
                        return 1
            return -1
        except:
```

```
            return -1

    # 11. NonStdPort
    def NonStdPort(self):
        try:
            port = self.domain.split(":")
            if len(port)>1:
                return -1
            return 1
        except:
            return -1

    # 12. HTTPSDomainURL
    def HTTPSDomainURL(self):
        try:
            if 'https' in self.domain:
                return -1
            return 1
        except:
            return -1

    # 13. RequestURL
    def RequestURL(self):
        try:
            for img in self.soup.find_all('img', src=True):
                dots = [x.start(0) for x in re.finditer('\.', img['src'])]
                if self.url in img['src'] or self.domain in img['src'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            for audio in self.soup.find_all('audio', src=True):
                dots = [x.start(0) for x in re.finditer('\.', audio['src'])]
                if self.url in audio['src'] or self.domain in audio['src'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            for embed in self.soup.find_all('embed', src=True):
                dots = [x.start(0) for x in re.finditer('\.', embed['src'])]
                if self.url in embed['src'] or self.domain in embed['src'] or len(dots) == 1:
```

```
                    success = success + 1
                i = i+1

            for iframe in self.soup.find_all('iframe', src=True):
                dots = [x.start(0) for x in re.finditer('\.', iframe['src'])]
                if self.url in iframe['src'] or self.domain in iframe['src'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            try:
                percentage = success/float(i) * 100
                if percentage < 22.0:
                    return 1
                elif((percentage >= 22.0) and (percentage < 61.0)):
                    return 0
                else:
                    return -1
            except:
                return 0
        except:
            return -1

    # 14. AnchorURL
    def AnchorURL(self):
        try:
            i,unsafe = 0,0
            for a in self.soup.find_all('a', href=True):
                if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto" in a['href'].lower() or not (self.url in a['href'] or self.domain in a['href']):
                    unsafe = unsafe + 1
                i = i + 1

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

```
            except:
                return -1

        except:
            return -1

    # 15. LinksInScriptTags
    def LinksInScriptTags(self):
        try:
            i,success = 0,0

            for link in self.soup.find_all('link', href=True):
                dots = [x.start(0) for x in re.finditer('\.', link['href'])]
                if self.url in link['href'] or self.domain in link['href'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            for script in self.soup.find_all('script', src=True):
                dots = [x.start(0) for x in re.finditer('\.', script['src'])]
                if self.url in script['src'] or self.domain in script['src'] or len(dots) == 1:
                    success = success + 1
                i = i+1

            try:
                percentage = success / float(i) * 100
                if percentage < 17.0:
                    return 1
                elif((percentage >= 17.0) and (percentage < 81.0)):
                    return 0
                else:
                    return -1
            except:
                return 0
        except:
            return -1

    # 16. ServerFormHandler
    def ServerFormHandler(self):
        try:
```

```
            if len(self.soup.find_all('form', action=True))==0:
                return 1
            else :
                for form in self.soup.find_all('form', action=True):
                    if form['action'] == "" or form['action'] == "about:blank":
                        return -1
                    elif self.url not in form['action'] and self.domain not in form['action']:
                        return 0
                    else:
                        return 1
        except:
            return -1

    # 17. InfoEmail
    def InfoEmail(self):
        try:
            if re.findall(r"[mail\(\)|mailto:?]", self.soap):
                return -1
            else:
                return 1
        except:
            return -1

    # 18. AbnormalURL
    def AbnormalURL(self):
        try:
            if self.response.text == self.whois_response:
                return 1
            else:
                return -1
        except:
            return -1

    # 19. WebsiteForwarding
    def WebsiteForwarding(self):
        try:
            if len(self.response.history) <= 1:
                return 1
            elif len(self.response.history) <= 4:
```

```
                return 0
            else:
                return -1
        except:
             return -1

    # 20. StatusBarCust
    def StatusBarCust(self):
        try:
            if re.findall("<script>.+onmouseover.+</script>", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1

    # 21. DisableRightClick
    def DisableRightClick(self):
        try:
            if re.findall(r"event.button ?== ?2", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1

    # 22. UsingPopupWindow
    def UsingPopupWindow(self):
        try:
            if re.findall(r"alert\(", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1

    # 23. IframeRedirection
    def IframeRedirection(self):
        try:
```

```
            if re.findall(r"[<iframe>|<frameBorder>]", self.response.text):
                return 1
            else:
                return -1
        except:
             return -1

    # 24. AgeofDomain
    def AgeofDomain(self):
        try:
            creation_date = self.whois_response.creation_date
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            today  = date.today()
            age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
            if age >=6:
                return 1
            return -1
        except:
            return -1

    # 25. DNSRecording
    def DNSRecording(self):
        try:
            creation_date = self.whois_response.creation_date
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            today  = date.today()
            age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
            if age >=6:
                return 1
```

```
            return -1
        except:
            return -1

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

    # 27. PageRank
    def PageRank(self):
        try:
            prank_checker_response = requests.post("https://www.checkpagerank.net/index.php", {"name": self.domain})

            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


    # 28. GoogleIndex
    def GoogleIndex(self):
        try:
            site = search(self.url, 5)
            if site:
                return 1
            else:
                return -1
        except:
            return 1

    # 29. LinksPointingToPage
```

```
    def LinksPointingToPage(self):
        try:
            number_of_links = len(re.findall(r"<a href=", self.response.text))
            if number_of_links == 0:
                return 1
            elif number_of_links <= 2:
                return 0
            else:
                return -1
        except:
            return -1

    # 30. StatsReport
    def StatsReport(self):
        try:
            url_match = re.search(
        'at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|sweddy\.com|myjino\.ru|96\.lt|ow\.ly', self.url)
            ip_address = socket.gethostbyname(self.domain)
            ip_match = re.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\.4
                                 '107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.151\.148\.108|107\.151\.148\.109|119\.28\.52\.61|54\.83\.43\.69|52
                                 '118\.184\.25\.86|67\.208\.74\.71|23\.253\.126\.58|104\.239\.157\.210|175\.126\.123\.219|141\.8\.224\.221|10\.10\.10\.10|43\.229\.108\.32|103\.
                                 '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|
                                 '34\.196\.13\.28|103\.224\.212\.222|172\.217\.4\.225|54\.72\.9\.51|192\.64\.147\.141|198\.200\.56\.183|23\.253\.164\.103|52\.48\.191\.26|52\.21
                                 '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\
            if url_match:
                return -1
            elif ip_match:
                return -1
            return 1
        except:
            return 1

    def getFeaturesList(self):
        return self.features
```

7.3 Data Base schema

https://drive.google.com/file/d/18PuytIZWvNQCnMypKdaQjAqqQ0MRZ0uj/view?usp=sharing

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
URL Detection Page_TC_OO1	Functional	Home Page	Verify user is able to see the Landing Page when user can type the URL in the box		1.Enter URL and click go 2.Type the URL 3.Verify whether it is processing or not.	https://careereducation.smartinternz.com/Student/guided_project_workspace/32250	Should Display the Webpage	Working as expected	Pass		N		Meenakshi A
URL Detection Page_TC_OO2	UI	Home Page	Verify the UI elements is Responsive		1.Enter 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	https://careereducation.smartinternz.com/Student/guided_project_workspace/32250	Should Wait for Response and then gets Acknowledge	Working as expected	Pass		N		Subasri M
URL Detection Page_TC_OO3	Functional	Home page	Verify whether the link is legitimate or not		1.Enter URL and click go 2. Type or copy paste the URL 3. Check the website is legitimate or not 4. Observe the results	https://www.youtube.com/	User should observe whether the website is legitimate or not.	Working as expected	Pass		N		Shiva Keerthana R
URL Detection Page_TC_OO4	Functional	Home page	Verify user is able to access the legitimate website or not		1.Enter URL and click go 2. Type or copy paste the URL 3. Check the website is legitimate or not	https://careereducation.smartinternz.com/Student/guided_project_workspace/32250	Application should show that Safe Webpage or Unsafe.	Working as expected	Pass		N		Vasunthara D
URL Detection Page_TC_OO5	Functional	Login page	Testing the website with multiple URLs		1.Enter URL(https://shopenzer.com/) and click go 2.Click on My Account dropdown button 3.Enter Valid username/email in Email text box 4.Enter Invalid password in password text box 5.Click on login button	1. https://careereducation.smartinternz.com/Student/guided_project_workspace/32250 2. https://drive.google.com/drive/folders/1bhoGGJ14fdsZzPfoZzpwZktB-kWTfggK	User can able to identify the websites whether it is secure or not	Working as expected	Pass		N		Swathi P

8.2 USER ACCEPTANCE TESTING

1.Purpose of Document

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

2.Defect Analysis

This reportshows the numberof resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal

By Design	10	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	10	2	4	20	36
Not Reproduced	0	0	1	0	1
Skipped	0	0	0	0	0
Won't Fix	0	0	2	1	3
Totals	23	9	12	25	60

3.Test Case Analysis

This report shows the numberof test cases that have passed, failed,and untested

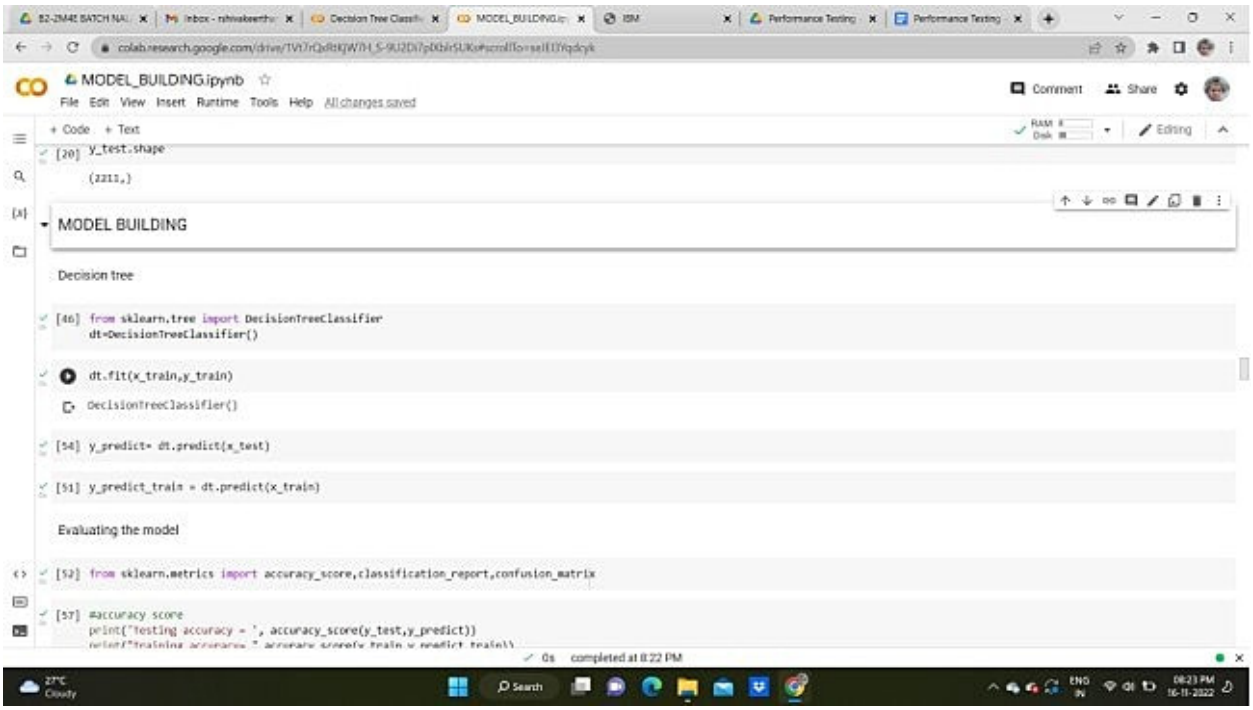
Section	TotalCases	Not Tested	Fail	Pass
Print Engine	10	0	0	10
Client Application	50	0	0	50
Security	5	0	0	4
Outsource Shipping	3	0	0	3
Exception Reporting	10	0	0	9
Final ReportOutput	10	0	0	10
Version Control	4	0	0	4

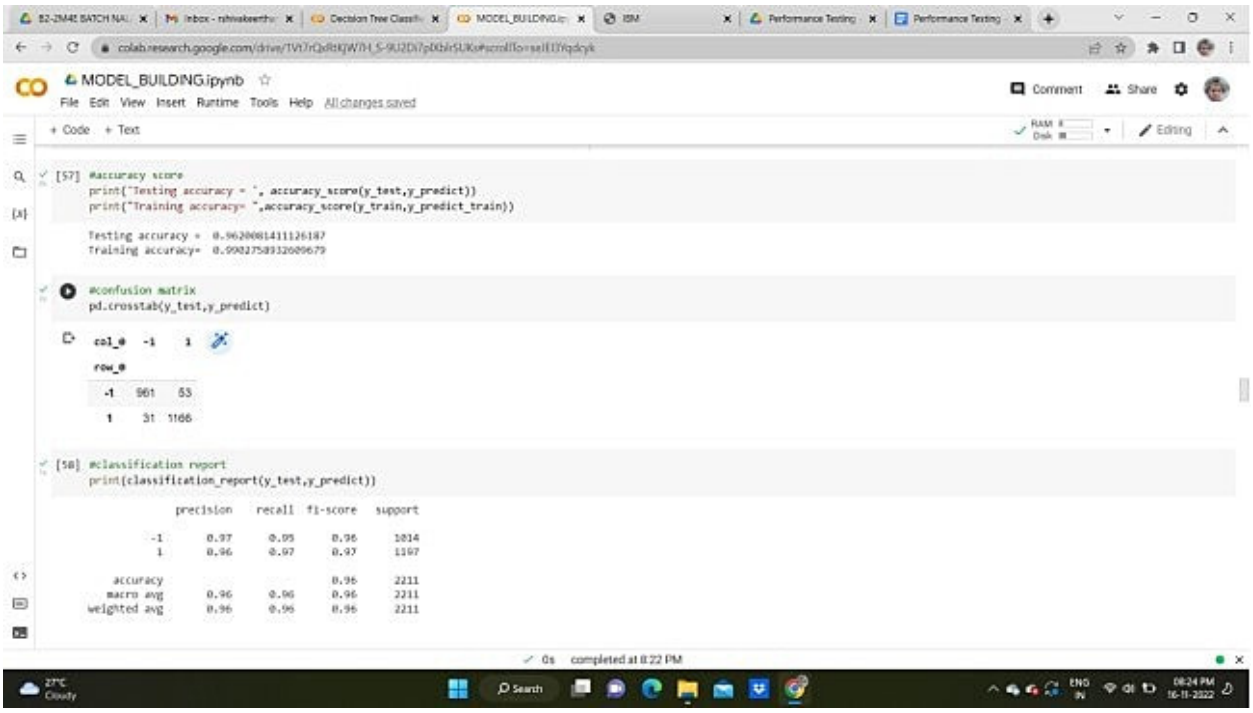
9.RESULTS

9.1 PERFORMANCE METRICS

1.Metrics

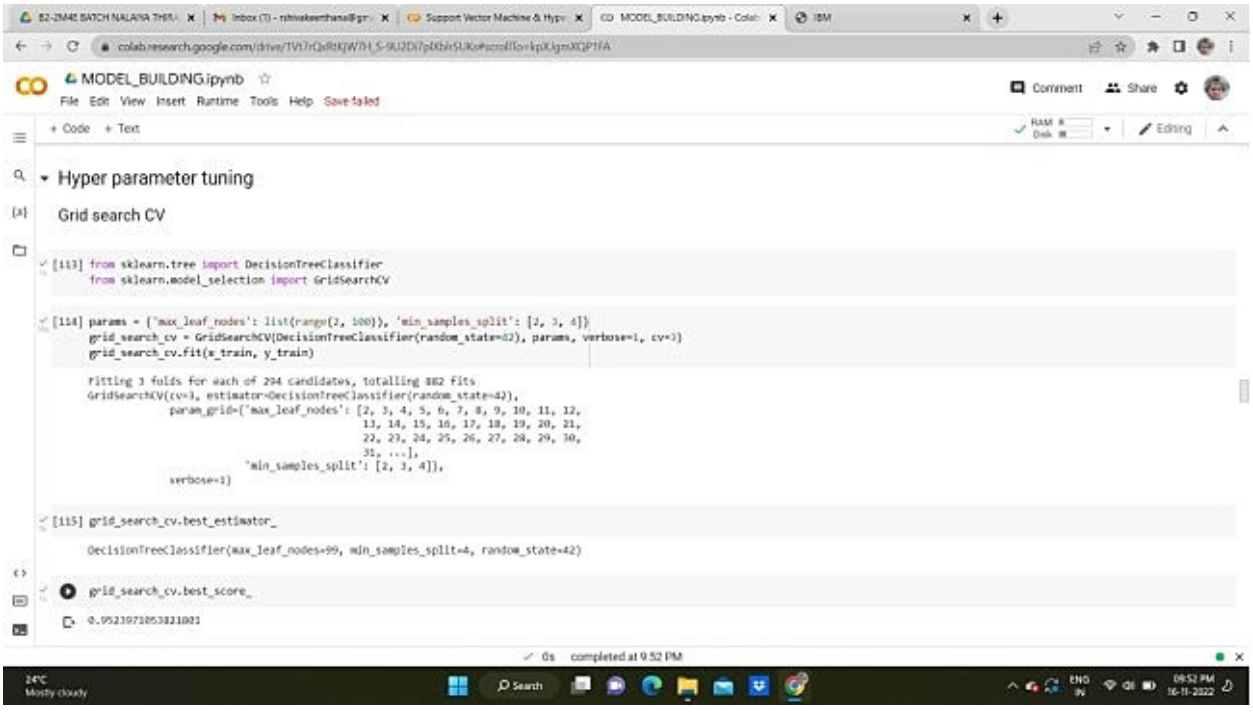
Classification Model: Confusion Matrix - , Accuracy Score- & Classification Report -

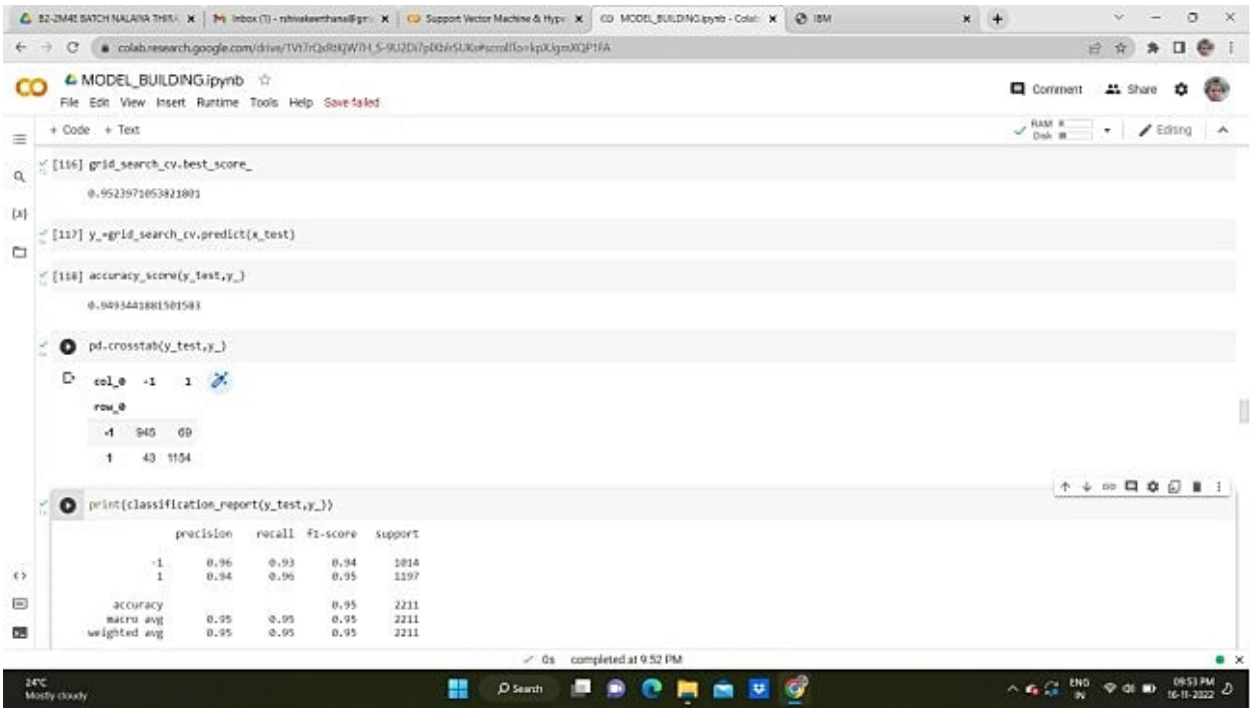




2.Tune the Model

Hyperparameter Tuning





10.ADVANTAGES AND DISADVANTAGES

Advantages

1. This system can be used by many E-commerce or other websites in order to have good customer relationship.
2. User can make online payment securely.
3. Decision Tree algorithm used in this system provides better performance as compared to other traditional classifications algorithms.
4. With the help of this system user can also visit any website without hesitation.

Disadvantages

1. If Internet connection fails, this system won’t work.

2. All websites related data will be stored in one place.

11.CONCLUSION

Thus to summarize, we have seen how phishing is a huge threat to the security and safety of the web and how phishing detection is an important problem domain. We have reviewed some of the traditional approaches to phishing detection; namely blacklist and heuristic evaluation methods, and their drawbacks. We have tested two machine learning algorithms on the Phishing Websites Dataset and reviewed their results. We then selected the best algorithm based on its performance and built a HTML Web page where the URL can be tested. The web page is user friendly . We have detected phishing websites using Decision Tree algorithm with an accuracy of 96.2%.

12.FUTURE SCOPE

Although the use of URL lexical features alone has been shown to result in high accuracy (96.2%), phishers have learned how to make predicting a URL destination difficult by carefully manipulating the URL to evade detection. Therefore, combining these features with others, such as host, is the most effective approach .For future enhancements, we intend to build the phishing detection system as a scalable web service which will incorporate online learning so that new phishing attack patterns can easily be learned and improve the accuracy of our models with better feature extraction.

13. APPENDIX

Source code

```
import pickle
import warnings

import numpy as np
import pandas as pd
from flask import Flask, render_template, request
from sklearn import metrics

warnings.filterwarnings('ignore')
from feature import FeatureExtraction

file = open("model.pkl","rb")
gbc = pickle.load(file)
file.close()


app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":

        url = request.form["url"]
        obj = FeatureExtraction(url)
        x = np.array(obj.getFeaturesList()).reshape(1,30)

        y_pred =gbc.predict(x)[0]
        #1 is safe
        #-1 is unsafe
        y_pro_phishing = gbc.predict_proba(x)[0,0]
        y_pro_non_phishing = gbc.predict_proba(x)[0,1]
        # if(y_pred ==1 ):
        pred = "It is {0:.2f} % safe to go ".format(y_pro_phishing*100)
        return render_template('index.html',xx =round(y_pro_non_phishing,2),url=url )
    return render_template("index.html", xx =-1)


if __name__ == "__main__":
    app.run(debug=True,port=2002)
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <center> <h1> WEB PHISHING DETECTION </h1> </center>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <style>
    body {
        background-color: lightcoral;
      }
    </style>
    <title>URL detection</title>
</head>
<body>
    <center>  <img class="image image-contain" src="https://mytechdecisions.com/wp-content/uploads/2020/02/email-phishing.jpg" alt="MDN logo" />

    <h2>URL BASED PHISHING DETECTION</h2>

    <br>
    <form action="/" method ="post">
    <label for="url" class="form__label">URL</label>
    <input type="text" class="form__input" name ='url' id="url" placeholder="Enter URL.." required="" />
    <br><br>
    <button class="button" role="button" >Check here</button>


    <h4><a href= {{ url }} target="_blank">{{ url }}</a></h4>
    <h3 id="prediction"></h3>
    <button class="button1" id="button1" role="button"  onclick="window.open('{{url}}')" target="_blank">Want to continue</button>
    </center>
```

```
34
35
36      <!-- JavaScript -->
37      <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
38          integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
39          crossorigin="anonymous"></script>
40      <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
41          integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
42          crossorigin="anonymous"></script>
43      <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
44          integrity="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
45          crossorigin="anonymous"></script>
46
47
48      <script>
49
50              let x = '{{xx}}';
51              let num = x*100;
52              if (0<=x && x<0.50){
53                  num = 100-num;
54              }
55              let txtx = num.toString();
56              if(x<=1 && x>=0.50){
57                  var label = "Prediction: Website is "+txtx +"% safe to use...";
58                  document.getElementById("prediction").innerHTML = label;
59                  document.getElementById("button1").style.display="block";
60              }
61              else if (0<=x && x<0.50){
62                  var label = "Website is "+txtx +"% unsafe to use..."
63                  document.getElementById("prediction").innerHTML = label ;
64                  document.getElementById("button1").style.display="block";
65              }
66
67      </script>
68  </body>
69  </html>
```

Github and project demo link

Github Link

https://github.com/IBM-EPBL/IBM-Project-32247-1660208781

Demo Link

https://drive.google.com/file/d/1T6kXFecY9bhwIYjSyuFczklyAxrQ4BU4/view?usp=share_link