CHAPTER 1

INTRODUCTION

Deceptive web pages are those, which contain content that can be used by attackers to exploit end-users. This includes web pages with phishing URLs; spam URLS, JavaScript malware scripts, Adware, and many more. Nowadays, it is becoming very difficult to detect such vulnerabilities due to the continuous development of new techniques for carrying out such attacks. In addition, not all the users are aware of the different type of exploits which attackers can take advantage of. Therefore, when there is vulnerability in a web page, which the user is not aware of, this tool will help him stay safe despite his lack of knowledge of the website. In addition, if the URL itself has signs of being a phishing URL, then the user will be protected from that website.

Python, being open source, along with the help of the variety of support libraries, simple to understand syntax, and abundant resources, proves to be the best option for implementing machine learning. The alternate methodology is to check the entered URL in the list of websites which are declared malicious by a trusted source. But the drawback of this method is that the list is non-exhaustive i.e. it increases every day. Also, because of such a large list, the latency time of the system will always increase which can frustrate the user. Hence, we use the Machine Learning approach, where the URL can be tested against a trained classifier.

A Google Chrome extension is a very good way of ensuring easy access to the tool for the user. Since Google Chrome is the most widely used web browser throughout the world and with its popularity only increasing month-by-month, it is the best option for implementing this tool to ensure the maximum outreach.

The goal is to ensure safe browsing irrespective of the website which the user wishes to visit. Even if the user decides to visit a deceptive website, measures will be taken to protect the user from being harmed.

CHAPTER 2

LITERATURE SURVEY

According to the survey conducted in “Survey on Malicious Web Pages Detection Technique”, the authors mention that the web attacks are on a rise. In the year 2012, each month noted a total of approximately 33,000 phishing attacks, which summed to a loss of \$687 million. With such a hike in the number of web attacks, there is a dire need of a system which prevents such types of webpage attacks. The user goes to a website unknowingly which might not be safe for the user and ends up either giving his credentials to other intruders and hackers or downloading malicious data. So, to prevent this type of attacks, a tool is needed which examines the URL entered by the user and checks if the website is malicious or not. In the paper "Identifying Vulnerable Websites by Analysis of Common Strings in Phishing URLs”, the authors mention about the rise in the phishing websites and the method they used to detect and prevent it. For the detection of a safe or a phishing webpage, they implemented Largest Common Substring (LCS) method. They prepared a database of phishing websites of their own and tested the LCS on the new website. In the paper “On URL Classification”, the authors mention about how URLs can be used to use the victim's computer resources for different attacks like phishing, denial of service. Also, they compared various methodologies including traditional ones and the recent trends in the field of machine learning. The results show that machine learning techniques are better for detection. So instead of using traditional techniques, we plan to integrate the concept of machine learning in our tool.

Now, as we have to examine whether the website is malicious or benign, we will have to extract the features of the website. In the paper “Feature Extraction Process: A Phishing Detection Approach”, the author talks about how the features can be extracted from a URL. The author finds 17 features which can be extracted from the URL based on which a URL can be declared as phishing or no. In our project, we plan to extract 22 features which will enable more accuracy in declaring a URL as malicious or benign. In the paper "Feature extraction and classification phishing websites based on URL", M. Aydin and N. Baykal used the feature extraction technique to form a feature matrix using which they classified the URL. They extracted 133 features and used only a subset of it which they considered as prominent.

They have not specified the reason for choosing the parameters using which they declare a website as malicious or no. They used different parameters and different algorithms to test the efficiency obtained. We plan to select a single algorithm and use all the parameters and features as given in the dataset we selected. In “A Comparison of Machine Learning Techniques for Phishing Detection”, Abu-Nimeh et al have done a comparative study of six different classifiers to find which classifier works the best. They have showed that Random Forest outperforms other classifiers by having the lowest error rate.

In the paper "Detection of phishing URLs using machine learning techniques", the author discusses about the rise of phishing websites and give techniques to extract features and implement machine learning algorithms to classify the same. They have extracted features like traffic rank details, lexical features, page rank etc. They have presented a study of different machine learning algorithms. A fixed result showing the best algorithm is not done in the paper, we will give statistical analysis of all the algorithms and even the accuracy of the chosen algorithm to prove the result.

As the technology advances, the number of possible malicious attacks would also increase. It would be impossible to prevent all the new malicious and phishing websites using the traditional methodology of storing the list of malicious URLs and checking directly from the database. So, “Malicious Web Content Detection using Machine learning” aims at improvising the traditional methodology by adding machine learning for the same. The paper uses four different classification algorithms to detect Dynamic HTML malicious codes and states that Boosted Decision Tree gives the best output. The prototype can determine whether webpage is malicious or not, but cannot block or prevent the malicious content. Now, there can be an instance that the URL might be benign but it may contain certain JavaScript code or iframe which will lead to download of malicious content. So, taking this possibility in consideration, we will scrape the entire webpage to detect malicious contents even if the website is rendered safe by the algorithm.

2.1 EXISTING SYSTEMS

- **Phish Detector** - A powerful extension to detect phishing attacks in Online-Banking web sites. It is a rule-based system that analyses the webpage content to identify phishing attacks. Phish Detector has the ability to detect online-banking phishing scams

quickly with zero false negative alarm. For accurate result, it is recommended to use this extension for your Online-Banking web pages only. This extension works only for banking websites and does not take into account the other domains.

- **Netcraft Extension** - The Netcraft Extension is a tool allowing easy lookup of information relating to the sites you visit and providing protection from Phishing. The Netcraft anti-phishing community is effectively a giant neighborhood watch scheme, empowering the most alert and most expert members to defend everyone within the community. As soon as the first recipients of a phishing mail report it, we can block it for all users of the extension providing an additional level of protection from Phishing. This extension waits for the first victim to report and accordingly notifies the other users for the same.
- **Cascaded Phish Detector** - The online cascaded phish detector is composed of a client-side component and a server-side component. The client side is implemented as a Chrome extension, which injects content script to web pages and extracts the corresponding HTML DOMs. This extension only considers the HTML DOMs for identifying the phishing components and no other parameters.

2.2 PROPOSED SYSTEM METHADOLOGY

Phishing attacks being on a rise, using a direct search from the phishing website database is not enough. To provide protection against new attacks, machine learning provides the best alternative for the same. We used the UCI Dataset of Phishing Website to train the classifier. Later, whenever a user enters the URL, the features are extracted and the URL is tested on the trained classifier to obtain the result. Following were the major steps involved in the implementation:

2.2.1 Obtaining Datasets

The dataset was obtained from the UCI - Machine Learning Repository which contains the Phishing Web Site Dataset. This dataset is composed of 11055 entries of websites which are classified as phishing and benign. These entries each have 30 features of the website used.

2.2.2 Feature Selection

From the dataset, out of the 30 features present, it was infeasible to extract all the features. This is because many features used some standard databases which are not accessible to us. Also, extracting some of the features seemed not possible as they demanded the extraction of data from the server of the website, which is not possible. Hence, we narrowed down our dataset to contain 22 features.

2.2.3 Choosing Classification Algorithm

For classifying the URL entered, as either safe or malicious, we considered the following algorithms:

- **K-Nearest Neighbors (kNN)** - kNN algorithm can be used for both classification as well as regression problems. However, mostly it is used for classification problems. 'k' in the kNN algorithm stands for the number of nearest neighbors we wish to take vote from. When predicting for a new data sample, this algorithm will run a search on the training dataset to find the closest k-samples. The predicted class of those similar samples is then found out and the summary of that is given out as the class label for this new data sample. Also, for different types of data, the similarity measure varies. The similarity measure for real-valued data can be Euclidean distance, whereas for other data like categorical or binary, hamming distance is helpful.
- **Support Vector Machines (SVM)** - They are supervised learning models used for both classification and regression. The SVM algorithm uses a dataset where the input samples are divided into two classes with labels either 0 or 1. The methodology includes finding a line (in two- dimension space) or a plane (in multi-dimension space) also called as a hyperplane which will most efficiently separate the two classes.

  $B0 + (B1 * X1) + (B2 * X2) = 0$ (1)

  In equation (1), the coefficients (B1 and B2) give the slope of the line and the intercept (B0) is calculated by the algorithm. Also, X1 and X2 are the two input data points. By plugging in these data points into the line equation, you can calculate whether a new point is on which side of the line and the class will be predicted.
- **Random Forest** - This algorithm makes use of decision trees for classification. It creates multiple decision trees at the time of training. Random Forests are a combination of tree predictors. The formation of each tree is based on the values of a

random vector sampled independently and each tree in the forest has the same distribution. The main principle is that a group of “weak learners” can combine and form a “strong learner”. Random Forests do not over fit data due to the law of large numbers and therefore are a wonderful tool for making predictions. When the right kind of randomness is introduced, they become accurate classifiers and repressors. Single decision trees often have high variance or high bias. Random Forests attempt to alleviate this problem by taking the average thereby finding a natural balance between the two extremes. Due to the fact that Random Forests have less number of parameters that can be tuned and that they can be used directly with default parameter settings, they are a simple tool to use without having a model or to produce a reasonable model in a fast and efficient manner.

- **LightGBM** - LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

  Faster training speed and higher efficiency.

  Lower memory usage.

  Better accuracy.

  Support of parallel and GPU learning.

  Capable of handling large-scale data.

  LightGBM uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage. LightGBM grows trees leaf-wise (best-first). It will choose the leaf with max delta loss to grow. Holding leaf fixed, leaf-wise algorithms tend to achieve lower loss than level-wise algorithms. Leaf-wise may cause over-fitting when data is small, so LightGBM includes the max_depth parameter to limit tree depth. However, trees still grow leaf-wise even when max_depth is specified.

- **Artificial Neural Network** - Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not necessarily identical to, the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analysing example images that have been manually labelled as "cat" or "no cat" and using the results to identify cats in other images. They do this

without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

For the comparative study of these algorithms, we not only consider the accuracy, but also other metrics which are used to select the best classifier. We calculated the accuracy, precision, recall, F1-Score for our dataset for all the algorithms. As per the result of these metrics, we got the best scores when the RF algorithm was used. Hence, we decided to train the classifier with the RF Algorithm. Below is the screenshot depicting the results of all the algorithms with the different performance metrics.

2.2.4 Google Chrome Extension

Extensions are add-ons to the browser which help in adding more features and making browser usage easier for the user. Browsers typically allow a variety of extensions, including user interface modifications, ad blocking, and cookie management. So, for our case, when a user enters the URL of the website, we display whether the URL entered is benign or deceptive using this Chrome Extension. HTML, JavaScript, CSS are used for coding a Google Chrome Extension.

2.3 DATASET FEATURES

One of the challenges faced by our research was the unavailability of reliable training datasets. In fact, this challenge faces any researcher in the field. However, although plenty of articles about predicting phishing websites using data mining techniques have been disseminated these days, no reliable training dataset has been published publicly, maybe because there is no agreement in literature on the definitive features that characterize phishing websites, hence it is difficult to shape a dataset that covers all possible features.

In this, we shed light on the important features that have proved to be sound and effective in predicting deceptive websites. In addition, we proposed some new features, experimentally assign new rules to some well-known features and update some other features.

2.3.1. Address Bar based Features

1. Using the IP Address

If an IP address is used as an alternative of the domain name in the URL, such as "http://125.98.3.123/fake.html", users can be sure that someone is trying to steal their personal information. Sometimes, the IP address is even transformed into hexadecimal code as shown in the following link "http://0x58.0xCC.0xCA.0x62/2/paypal.ca/index.html".

$$\text{Rule: IF}\begin{cases}\text{If The Domain Part has an IP Address} \rightarrow \text{Phishing}\\ \text{Otherwise} \rightarrow \text{Legitimate}\end{cases}$$

2. Long URL to Hide the Suspicious Part

Phishers can use long URL to hide the doubtful part in the address bar. For example: http://federmacedoadv.com.br/3f/aze/ab51e2e319e51502f416dbe46b773a5e/?cmd=_home&dispatch=11004d58f5b74f8dc1e7c2e8dd4105e811004d58f5b74f8dc1e7c2e8dd4105e8@phishing.website.html To ensure accuracy of our study, we calculated the length of URLs in the dataset and produced an average URL length. The results showed that if the length of the URL is greater than or equal 54 characters then the URL classified as phishing. By reviewing our dataset, we were able to find 1220 URLs lengths equals to 54 or more which constitute 48.8% of the total dataset size.

$$\text{Rule: IF}\begin{cases}\text{URL length} < 54 \rightarrow \text{feature} = \text{Legitimate}\\ \text{else if URL length} \geq 54 \text{ and } \leq 75 \rightarrow \text{feature} = \text{Suspicious}\\ \text{otherwise} \rightarrow \text{feature} = \text{Phishing}\end{cases}$$

3. Using URL Shortening Services "TinyURL"

URL shortening is a method on the "World Wide Web" in which a URL may be made considerably smaller in length and still lead to the required webpage. This is accomplished by means of an "HTTP Redirect" on a domain name that is short, which links to the webpage that

has a long URL. For example, the URL “http://portal.hud.ac.uk/” can be shortened to “bit.ly/19DXSk4”.

$$\text{Rule: IF} \begin{cases} \text{TinyURL} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

4. URL’s having “@” Symbol

Using “@” symbol in the URL leads the browser to ignore everything preceding the “@” symbol and the real address often follows the “@” symbol.

$$\text{Rule: IF} \begin{cases} \text{Url Having @ Symbol} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

5. Redirecting using “//”

The existence of “//” within the URL path means that the user will be redirected to another website. An example of such URL’s is: “http://www.legitimate.com//http://www.phishing.com”. We examine the location where the “//” appears. We find that if the URL starts with “HTTP”, that means the “//” should appear in the sixth position. However, if the URL employs “HTTPS” then the “//” should appear in seventh position.

$$\text{Rule: IF} \begin{cases} \text{ThePosition of the Last Occurrence of "//" in the URL} > 7 \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

6. Adding Prefix or Suffix Separated by (-) to the Domain

The dash symbol is rarely used in legitimate URLs. Phishers tend to add prefixes or suffixes separated by (-) to the domain name so that users feel that they are dealing with a legitimate webpage. For example, http://www.Confirme-paypal.com/.

$$\text{Rule: IF} \begin{cases} \text{Domain Name Part Includes } (-) \text{ Symbol} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

7. Sub Domain and Multi Sub Domains

Let us assume we have the following link: http://www.hud.ac.uk/students/. A domain name might include the country-code top-level domains (ccTLD), which in our example is "uk". The "ac" part is shorthand for "academic", the combined "ac.uk" is called a second-level domain (SLD) and "hud" is the actual name of the domain. To produce a rule for extracting this feature, we firstly have to omit the (www.) from the URL which is in fact a sub domain in itself. Then, we have to remove the (ccTLD) if it exists. Finally, we count the remaining dots. If the number of dots is greater than one, then the URL is classified as "Suspicious" since it has one sub domain. However, if the dots are greater than two, it is classified as "Phishing" since it will have multiple sub domains. Otherwise, if the URL has no sub domains, we will assign "Legitimate" to the feature.

$$\text{Rule: IF} \begin{cases} \text{Dots In Domain Part} = 1 \rightarrow \text{Legitimate} \\ \text{Dots In Domain Part} = 2 \rightarrow \text{Suspicious} \\ \text{Otherwise} \rightarrow \text{Phishing} \end{cases}$$

8. Domain Registration length

Based on the fact that a phishing website lives for a short period of time, we believe that trustworthy domains are regularly paid for several years in advance. In our dataset, we find that the longest fraudulent domains have been used for one year only.

$$\text{Rule: IF} \begin{cases} \text{Domains Expires on} \leq 1 \text{ years} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

9. Favicon

A favicon is a graphic image (icon) associated with a specific webpage. Many existing user agents such as graphical browsers and newsreaders show favicon as a visual reminder of the website identity in the address bar. If the favicon is loaded from a domain other than that shown in the address bar, then the webpage is likely to be considered a Phishing attempt.

$$\text{Rule: IF} \begin{cases} \text{Favicon Loaded From External Domain} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

10. The Existence of "HTTPS" Token in the Domain Part of the URL

The phishers may add the "HTTPS" token to the domain part of a URL in order to trick users. For example, http://https-www-paypal-it-webapps-mpp-home.soft-hair.com/.

$$\text{Rule: IF}\begin{cases}\text{Using HTTP Token in Domain Part of The URL} \rightarrow \text{Phishing}\\ \text{Otherwise} \rightarrow \text{Legitimate}\end{cases}$$

2.3.2 Abnormal Based Features

1. Request URL

Request URL examines whether the external objects contained within a webpage such as images, videos and sounds are loaded from another domain. In legitimate webpages, the webpage address and most of objects embedded within the webpage are sharing the same domain.

$$\text{Rule: IF}\begin{cases}\%\text{ of Request URL} < 22\% \rightarrow \text{Legitimate}\\ \%\text{of Request URL} \geq 22\% \text{ and } 61\% \rightarrow \text{Suspicious}\\ \text{Otherwise} \rightarrow \text{feature} = \text{Phishing}\end{cases}$$

2. URL of Anchor

An anchor is an element defined by the <a> tag. This feature is treated exactly as "Request URL". However, for this feature we examine:

1. If the <a> tags and the website have different domain names. This is similar to request URL feature.
2. If the anchor does not link to any webpage, e.g.:
   A. <a href="#">
   B. <a href="#content">
   C. <a href="#skip">
   D. <a href="JavaScript ::void(0)">

$$\text{Rule: IF}\begin{cases}\%\text{ of URL Of Anchor} < 31\% \rightarrow \text{Legitimate}\\ \%\text{ of URL Of Anchor} \geq 31\% \text{ And} \leq 67\% \rightarrow \text{Suspicious}\\ \text{Otherwise} \rightarrow \text{Phishing}\end{cases}$$

3. Links in <Meta>, <Script> and <Link> tags

Given that our investigation covers all angles likely to be used in the webpage source code, we find that it is common for legitimate websites to use <Meta> tags to offer metadata about the HTML document; <Script> tags to create a client side script; and <Link> tags to retrieve other web resources. It is expected that these tags are linked to the same domain of the webpage.

$$\text{Rule: IF}\begin{cases} \%\text{ of Links in tags} < 17\% \rightarrow \text{Legitimate} \\ \%\text{ of Links in tags} \geq 17\% \text{ And} \leq 81\% \rightarrow \text{Suspicious} \\ \text{Otherwise} \rightarrow \text{Phishing} \end{cases}$$

4. Server Form Handler (SFH)

SFHs that contain an empty string or “about:blank” are considered doubtful because an action should be taken upon the submitted information. In addition, if the domain name in SFHs is different from the domain name of the webpage, this reveals that the webpage is suspicious because the submitted information is rarely handled by external domains.

$$\text{Rule: IF}\begin{cases} \text{SFH is "about: blank" Or Is Empty} \rightarrow \text{Phishing} \\ \text{SFH Refers To A Different Domain} \rightarrow \text{Suspicious} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

5. Submitting Information to Email

Web form allows a user to submit his personal information that is directed to a server for processing. A phisher might redirect the user’s information to his personal email. To that end, a server-side script language might be used such as “mail()” function in PHP. One more client-side function that might be used for this purpose is the “mailto:” function.

$$\text{Rule: IF}\begin{cases} \text{Using "mail()" or "mailto:" Function to Submit User Info} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

6. Abnormal URL

This feature can be extracted from WHOIS database. For a legitimate website, identity is typically part of its URL.

$$\text{Rule: IF} \begin{cases} \text{The Host Name Is Not Included In URL} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

2.3.3 HTML and JavaScript based Features

1. IFrame Redirection

IFrame is an HTML tag used to display an additional webpage into one that is currently shown. Phishers can make use of the "iframe" tag and make it invisible i.e. without frame borders. In this regard, phishers make use of the "frameBorder" attribute which causes the browser to render a visual delineation.

$$\text{Rule: IF} \begin{cases} \text{Using iframe} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

2.3.4 Domain based Features

1. Age of Domain

This feature can be extracted from WHOIS database (Whois 2005). Most phishing websites live for a short period of time. By reviewing our dataset, we find that the minimum age of the legitimate domain is 6 months.

$$\text{Rule: IF} \begin{cases} \text{Age Of Domain} \geq 6 \text{ months} \rightarrow \text{Legitimate} \\ \text{Otherwise} \rightarrow \text{Phishing} \end{cases}$$

2. DNS Record

For phishing websites, either the claimed identity is not recognized by the WHOIS database (Whois 2005) or no records founded for the hostname (Pan and Ding 2006). If the DNS record is empty or not found then the website is classified as "Phishing", otherwise it is classified as "Legitimate".

$$\text{Rule: IF} \begin{cases} \text{no DNS Record For The Domain} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$$

3. Website Traffic

This feature measures the popularity of the website by determining the number of visitors and the number of pages they visit. However, since phishing websites live for a short period of time, they may not be recognized by the Alexa database (Alexa the Web Information Company., 1996). By reviewing our dataset, we find that in worst scenarios, legitimate websites ranked among the top 100,000. Furthermore, if the domain has no traffic or is not recognized by the Alexa database, it is classified as "Phishing". Otherwise, it is classified as "Suspicious".

$$\text{Rule: IF}\begin{cases}\text{Website Rank} < 100{,}000 \rightarrow \text{Legitimate} \\ \text{Website Rank} > 100{,}000 \rightarrow \text{Suspicious} \\ \text{Otherwise} \rightarrow \text{Phish}\end{cases}$$

4. Google Index

This feature examines whether a website is in Google's index or not. When a site is indexed by Google, it is displayed on search results (Webmaster resources, 2014). Usually, phishing webpages are merely accessible for a short period and as a result, many phishing webpages may not be found on the Google index.

$$\text{Rule: IF}\begin{cases}\text{Webpage Indexed by Google} \rightarrow \text{Legitimate} \\ \text{Otherwise} \rightarrow \text{Phishing}\end{cases}$$

5. Statistical-Reports Based Feature

Several parties such as PhishTank (PhishTank Stats, 2010-2012), and StopBadware (StopBadware, 2010-2012) formulate numerous statistical reports on phishing websites at every given period of time; some are monthly and others are quarterly. In our research, we used 2 forms of the top ten statistics from PhishTank: "Top 10 Domains" and "Top 10 IPs" according to statistical-reports published in the last three years, starting in January2010 to November 2012. Whereas for "StopBadware", we used "Top 50" IP addresses.

$$\text{Rule: IF}\begin{cases}\text{Host Belongs to Top Phishing IPs or Top Phishing Domains} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate}\end{cases}$$

2.4 ADVANTAGES OF THE PROPOSED SYSTEM

- Lightweight and can be implemented within browsers as add-ons.
- Highly accurate and, works in real time without any updates.
- Low complexity which is equivalent to single or multiple web search and string matching operations per URL.
- Low-storage complexity, nothing gets stored at client side.
- Dynamic training to increase the accuracy of classification.
- The use of new classification features and machine learning algorithms can be implemented in the server without effecting client application hence improving accuracy, making the scheme adaptable.
- Equires less feature extraction from websites in the form of texts.
- Helps in detecting phishing websites before users end up giving his credentials.

CHAPTER 3

REQUIREMENTS

3.1 Hardware Requirements:

- High Performance GPUs for training.
- Laptop

3.2 Software Requirements:

- Google Chrome Browser (Developer Mode)
- Flask Application Server
- Active Internet Connection
- Python Interpreter
- Jupyter Notebook
- Python Based ML Classification Algorithm Libraries (Scikit-learn)
- Python Based Artificial Neural Network Library (Keras with Tensorflow backend)
- Python Based Web Scrapping Libraries (BeautifulSoup)
- Python Based Model Exporting/Importing (Pickling/Unpickling) Libraries (joblib)
- Integrated Development Environment (VS Code)

CHAPTER 4

SYSTEM ANALYSIS

4.1 Introduction

Phishing is the fraudulent attempt to obtain sensitive information such as usernames, passwords and credit card details by disguising oneself as a trustworthy entity in an electronic communication. Typically carried out by deceptive webpages, it often directs users to enter personal information at a fake website which matches the look and feel of the legitimate site. Phishing is an example of social engineering techniques being used to deceive users. Users are often lured by communications purporting to be from trusted parties such as social web sites, auction sites, banks, online payment processors or IT administrators. Attempts to deal with phishing incidents include legislation, user training, public awareness, and technical security measures (the latter being due to phishing attacks frequently exploiting weaknesses in current web security).

To overcome these attacks in the best way is to make use of a chrome extension since that the most widely used web browser in the world. A chrome extension is built so that whenever a user visits any webpage the extension will extract the current URL and send that to server through HTTP POST Request with the API key and at the server side the feature extraction will be done for respective webpage of the received URL and prediction will be made using pre-loaded artificial neural network model & the prediction will sent back to the client application through JSON response & client will notify the user about the webpage whether it is deceptive or not.

Hence it will ensure that the user is having a safe browsing experience & will not give out the credentials to hackers.

4.2 Feature Selection

In the process of feature selection, not all features from the web page could be extracted since some features require access to their main database. 22 features from the dataset out of the 30 have been included for training the model and all these check to the constraints and see

if it satisfies the conditions in our trained model. Features of the category address bar based, abnormal based, HTML and JavaScript based & domain-based features have been included. Based on these sub-category's features are selected and a machine learning model is trained using this data to obtain high classification accuracy.

4.3 Data Collection and Training

The data is collected from the UCI - Machine Learning Repository which contains the Phishing Web Site Dataset. This dataset is composed of 11055 entries of websites which are classified as phishing and benign. These entries each have 30 features of the website used.

Training of machine learning model process includes

- Data collection
- Pre-processing & Feature Selection
- Training Model using appropriate algorithm with tuned hyperparameters
- Exporting trained model to a pickle file

4.4 Monitoring and Phishing Detection

Our application-specific phishing detection aims to ensure the integrity of commonly used applications (e.g. web browser). After the data collection, feature selection and training process, the trained model will be deployed in the server. It can be accessed using the API through the chrome extension.

4.5 List of Modules

There are various important modules that serve as the core of the proposed system, A list of these prominent modules are mentioned below:

4.5.1 Training ML Model

- Dataset
- Feature Selection
- Classification Algorithm

4.5.2 Production Environment

- Chrome Extension
- API Server
- Feature Extractor
- Predictor

4.6 Description of Modules

1. Dataset: The dataset is obtained from the UCI - Machine Learning Repository which contains the Phishing Web Site Dataset. This dataset is composed of 11055 entries of websites which are classified as phishing and benign. These entries each have 30 features of the website used.

2. Feature Selection: From the dataset, out of the 30 features present, it was infeasible to extract all the features. This is because many features used some standard databases which are not accessible to us. Also, extracting some of the features seemed not possible as they demanded the extraction of data from the server of the website, which is not possible. Hence, we narrowed down our dataset to contain 22 features.

3. Classification Algorithm: For classifying the URL as either safe or malicious, we considered the following three algorithms:

K-Nearest Neighbors (KNN) - KNN algorithm can be used for both classification as well as regression problems. However, mostly it is used for classification problems. Accuracy: 93.07400 %

Support Vector Machines (SVM) - They are supervised learning models used for both classification and regression. Accuracy: 93.54838 %

Random Forest (RF) - This algorithm makes use of decision trees for classification. It creates multiple decision trees at the time of training. Random Forests are a combination of tree predictors. Accuracy: 95.11005 %

Light Gradient Booster (LGB) - LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins. Accuracy: 95.635673 %

Sequential Artificial Neural Network (ANN) - An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. Accuracy: 96.20165 %

4. Chrome Extension: Extensions are add-ons to the browser which help in adding more features and making browser usage easier for the user. So, in our case, when a user visits any website, we display whether the page is benign or phishing using this Chrome Extension. HTML, JavaScript, jQuery, CSS, JSON are used for coding a Google Chrome Extension.

5. API Server: API is the Centralized Component of the application which connects the client-side Chrome Extension with the Feature Extractor and ML Predictor Residing at the server side. The request validation is done by using the key which is sent by the Chrome Extension as a part of HTTP POST Request Body.

6. Feature Extractor: Feature Extractor takes the URL from POST Request and Run multiple threads Extracting the feature of the URL and particular webpage That URL represents.

Such as

- URL Shortening service
- Having multiple subdomains
- HTTP or HTTPS token in domain name
- DNS Record
- Web Traffic
- Google Index etc.…

7. Predictor: Predictor is a loaded machine Learning model at Server (Done by loading .pkl file generated) It takes the features and predicts weather the URL (feature extracted from the url) is safe or not.

8. Scikit-learn: Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including the support vector machines, random forest and gradient boosting and is designed to interoperate with the python numerical and scientific libraries.

9. Keras: Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier

10. Tensorflow: It is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow is used to create large-scale neural networks with many layers. TensorFlow is mainly used for deep learning or machine learning problems such as Classification, Perception, Understanding, Discovering, Prediction and Creation.

11. urllib: urllib is a Python module that can be used for opening URLs. It defines functions and classes to help in URL actions.With Python you can also access and retrieve data from the internet like XML, HTML, JSON, etc. You can also use Python to work with this data directly

12. flask: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.[3] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

13. Visual studio Code: It is s a source-code editor developed by Microsoft for Windows, Linux and macOS.It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is highly

customizable, allowing users to change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The source code is free and open source and released under the permissive MIT License. The compiled binaries are freeware and free for private or commercial use.

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

CHAPTER 5

SYSTEM DESIGN

5.1 Architecture Design

A good architecture covers all crucial concerns of data, security and privacy concerns & of course a good architecture should address technical concerns in order to minimize the risk of instant project failure. In our design system we have two parts included

- Client side
- Server side

In the client side, a chrome browser extension is developed which has to be used by the user in order to detect the phishing site. Whenever the user enters the URL into the search bar or visits any page through navigation of the google chrome browser, the URL will be sent to server API using HTTP POST request with the API key.

On the server side the received key will be validated on successful key validation, the 22 features of the received URL is extracted & then using these features the prediction is made using the pre-loaded sequential artificial neural network model and once the prediction is made the result will be sent back to client application(chrome extension) through JSON response & the client application(chrome extension) will notify the user about the visited website whether it is safe or deceptive through alerts.

5.2 System Design

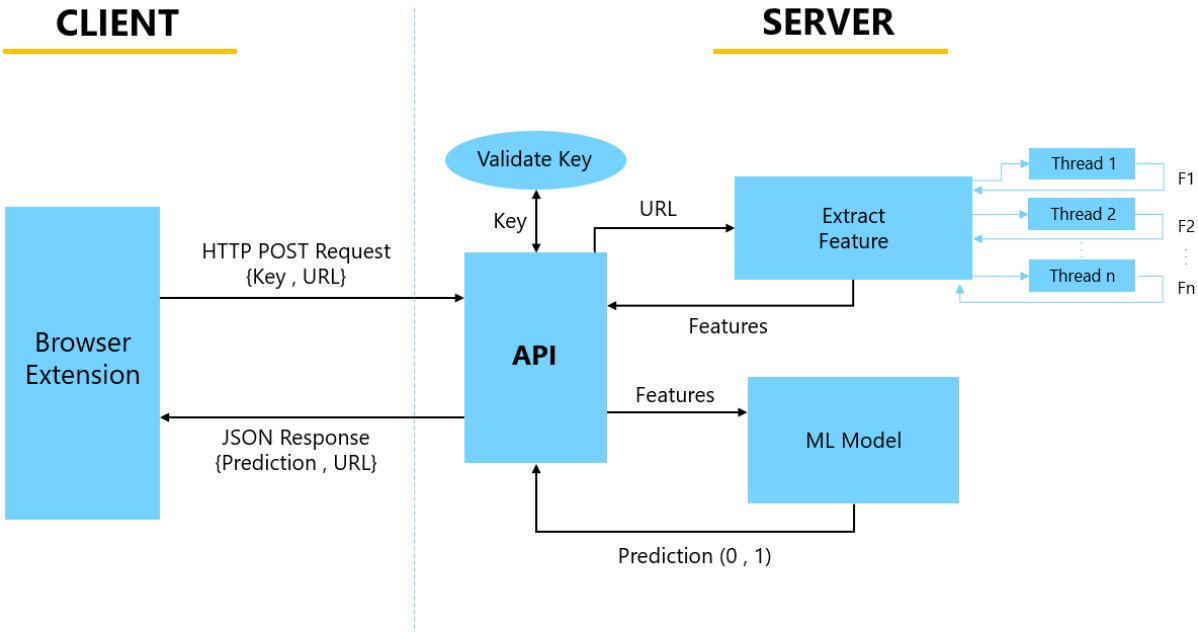

Fig 5.2.1 System Architecture

5.3 High Level Design

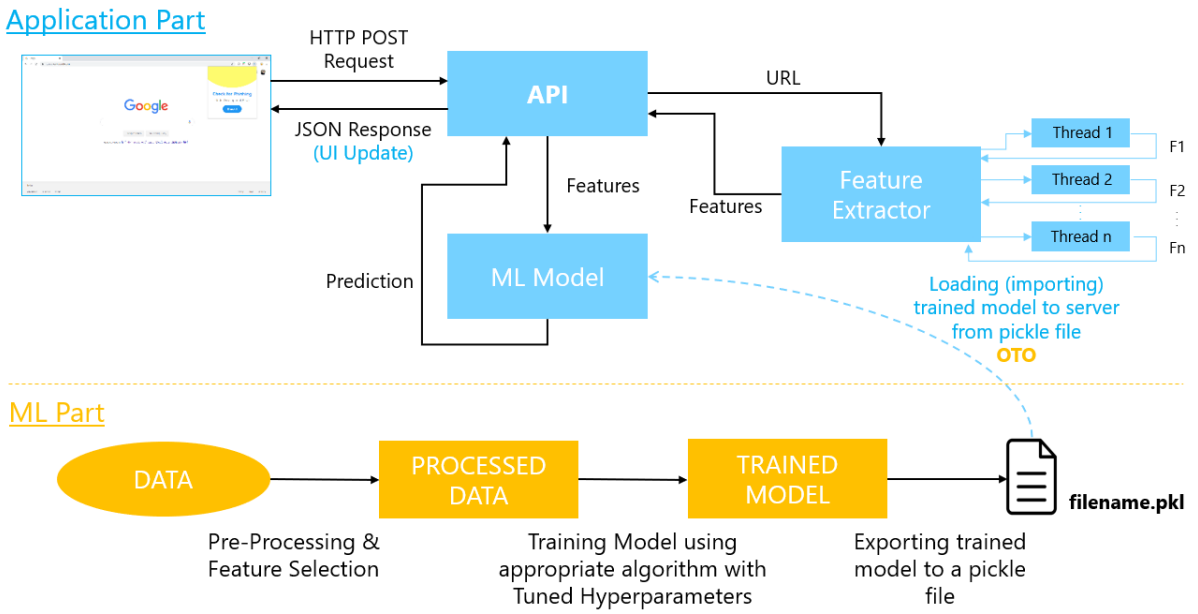

Fig 5.3.1 High Level Design

5.4 UML Diagram

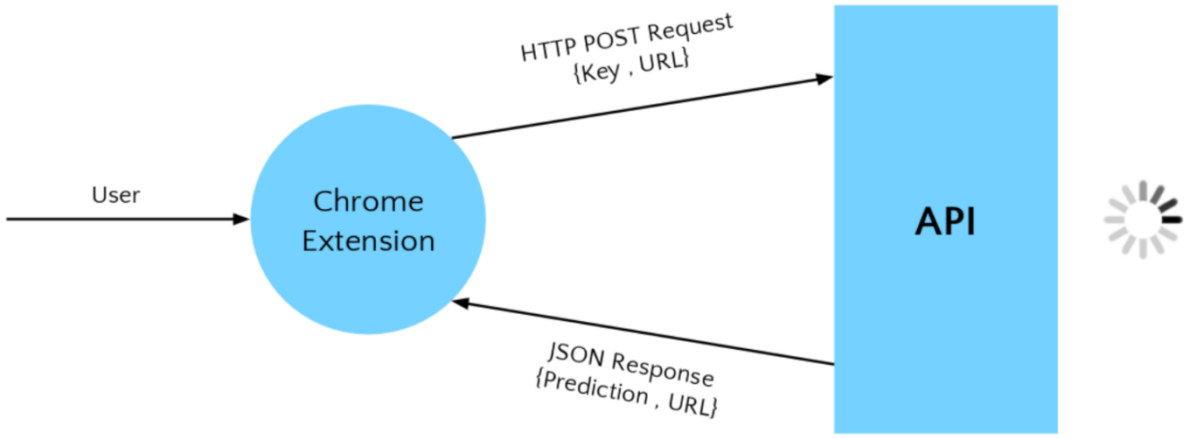

Fig 5.4.1 UML Design

5.5 Control Flow Diagram

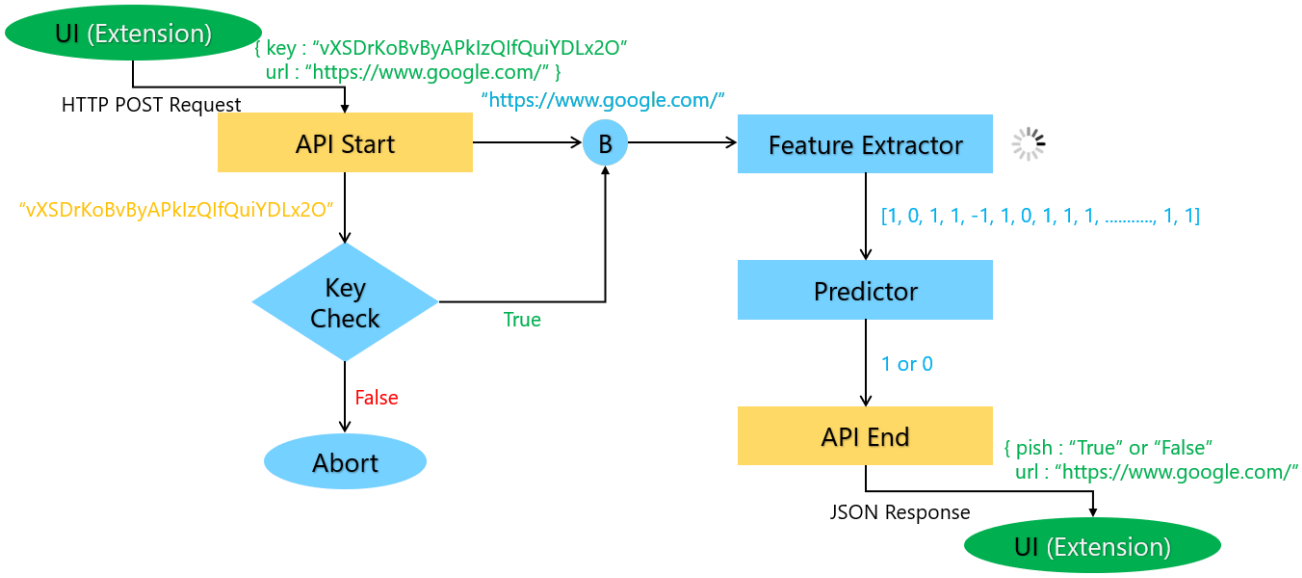

Fig 5.5.1 Control flow and working of the system

CHAPTER 6

IMPLEMENTATION

6.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

6.2 Flask Application Server (API)

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if

they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program. Flask is commonly used with MongoDB, which gives it more control over databases and history

6.3 Chrome Extension

Extensions are small software programs that customize the browsing experience. They enable users to tailor Chrome functionality and behavior to individual needs or preferences. They are built on web technologies such as HTML, JavaScript, and CSS.

An extension must fulfill a single purpose that is narrowly defined and easy to understand. A single extension can include multiple components and a range of functionality, as long as everything contributes towards a common purpose.

A screenshot of an extension's icon in the browser bar User interfaces should be minimal and have intent. They can range from a simple icon, such as the Google Mail Checker extension shown on the right, to overriding an entire page.

Extension files are zipped into a single .crx package that the user downloads and installs. This means extensions do not depend on content from the web, unlike ordinary web apps.

Extensions are distributed through the Chrome Developer Dashboard and published to the Chrome Web Store.

6.4 Artificial Neural Network

Artificial Neural Networks (ANN) is a supervised learning system built of a large number of simple elements, called neurons or perceptron’s. Each neuron can make simple decisions, and feeds those decisions to other neurons, organized in interconnected layers. Together, the neural network can emulate almost any function, and answer practically any question, given enough training samples and computing power. A “shallow” neural network has only three layers of neurons:

An input layer that accepts the independent variables or inputs of the model

One hidden layer

An output layer that generates predictions

A Deep Neural Network (DNN) has a similar structure, but it has two or more “hidden layers” of neurons that process inputs. Neural networks are able to tackle complex problems, deep learning networks are more accurate, and improve in accuracy as more neuron layers are added. Additional layers are useful up to a limit of 9-10, after which their predictive power starts to decline. Today most neural network models and implementations use a deep network of between 3-10 neuron layers.

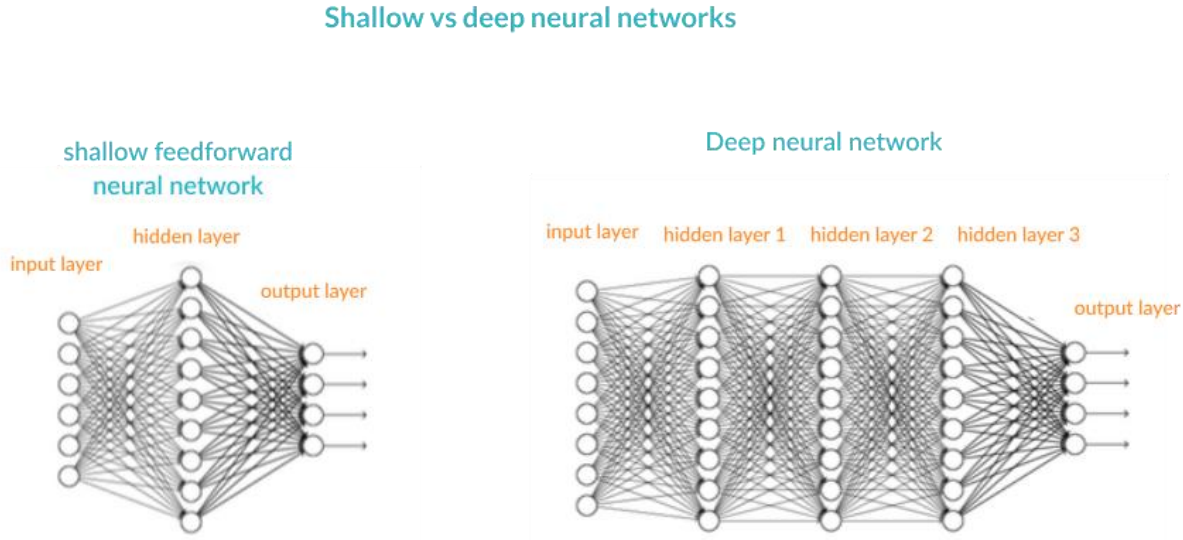

Fig 6.4.1 Shallow vs Deep Neural Network

Inputs

Source data fed into the neural network, with the goal of making a decision or prediction about the data. Inputs to a neural network are typically a set of real values; each value is fed into one of the neurons in the input layer.

Training Set

A set of inputs for which the correct outputs are known, used to train the neural network.

Outputs

Neural networks generate their predictions in the form of a set of real values or Boolean decisions. Each output value is generated by one of the neurons in the output layer.

Neuron/Perceptron

The basic unit of the neural network. Accepts an input and generates a prediction.

Activation Function

Each neuron accepts part of the input and passes it through the activation function. Common activation functions are sigmoid, TanH and ReLu. Activation functions help generate output values within an acceptable range, and their non-linear form is crucial for training the network.

Weight Space

Each neuron is given a numeric weight. The weights, together with the activation function, define each neuron's output. Neural networks are trained by fine-tuning weights, to discover the optimal set of weights that generates the most accurate prediction.

Forward Pass

The forward pass takes the inputs, passes them through the network and allows each neuron to react to a fraction of the input. Neurons generate their outputs and pass them on to the next layer, until eventually the network generates an output.

Error Function

Defines how far the actual output of the current model is from the correct output. When training the model, the objective is to minimize the error function and bring output as close as possible to the correct value.

Backpropagation

In order to discover the optimal weights for the neurons, we perform a backward pass, moving back from the network's prediction to the neurons that generated that prediction. This is called backpropagation. Backpropagation tracks the derivatives of the activation functions in each successive neuron, to find weights that brings the loss function to a minimum, which will generate the best prediction. This is a mathematical process called gradient descent.

Bias and Variance

When training neural networks, like in other machine learning techniques, we try to balance between bias and variance. Bias measures how well the model fits the training set—able to correctly predict the known outputs of the training examples. Variance measures how well the model works with unknown inputs that were not available during training. Another meaning of bias is a "bias neuron" which is used in every layer of the neural network. The bias neuron holds the number 1, and makes it possible to move the activation function up, down, left and right on the number graph.

Hyperparameters

A hyperparameter is a setting that affects the structure or operation of the neural network. In real deep learning projects, tuning hyperparameters is the primary way to build a network that provides accurate predictions for a certain problem. Common hyperparameters include the number of hidden layers, the activation function, and how many times (epochs) training should be repeated.

6.5 Code Implementation

6.5.1 Client: These two are two of the main modules of our system and these are the Chrome Extension as client which we use to make an api call to server with the url of the current page and get the response from the server and update the ui accordingly.

Source code for Client:

Manifest.json

```
{
  "manifest_version": 2,
  "name": "Phishing Webpage Detector",
  "description": "Under Development (Will be updated soon)",
  "version": "1.0",
  "browser_action": {
        "default_icon": "eye.png",
        "default_popup": "popup.html"
```

```
  },
 "background": {
  "scripts": ["background.js"],
  "persistent": false
 },
 "content_security_policy": "script-src 'self' https://ajax.googleapis.com; object-src 'self'",
 "permissions": [
  "activeTab",
  "tabs",
  "https://ajax.googleapis.com/",
  "tabs",
  "<all_urls>"  ]
}
```

Background.js

```
chrome.tabs.onUpdated.addListener(function(tabId, changeInfo, tab) {
        if(changeInfo.url.toString().startsWith("chrome")){ }
        else{
                chrome.tabs.executeScript({file: inject.js'});
                var tablink;
                tablink = changeInfo.url.toString();
                var xhr=new XMLHttpRequest();
                var key = "vXSDrKoBvByAPkIzQIfQuiYDLx2OvH7wIypN67Mn";
                var markup = "key="+key+"&url="+tablink;
                xhr.open("POST","http://127.0.0.1:5000/api/v2/pishcheck",false);
                xhr.setRequestHeader("Content-type","application/www-form-urlencoded");
                xhr.send(markup);
                var data = JSON.parse(xhr.responseText);
                if(data.pish=="True"){
                        chrome.tabs.executeScript({file: 'phish.js'});
                }
                else if(data.pish == "False"){
```

```
                chrome.tabs.executeScript({file: 'safe.js'});
            }
            else{
                alert("Oh ohh Unable to Predict");
            }
        }
});
```

Popup.js

```
function transfer(){
    var tablink;
    chrome.tabs.getSelected(null,function(tab) {
        tablink = tab.url;
        var xhr=new XMLHttpRequest();
        var key = "vXSDrKoBvByAPkIzQIfQuiYDLx2OvH7wIypN67Mn";
        var markup = "key="+key+"&url="+tablink;
        xhr.open("POST","http://127.0.0.1:5000/api/v2/pishcheck",false);
        xhr.setRequestHeader("Content-type","application/www-form-urlencoded");
        xhr.send(markup);
        var data = JSON.parse(xhr.responseText);
        if(data.pish=="True"){
            $("#info").text("Phishing Site");
            $("#info").css("color", "red");
            $("#image").css('background-image', 'url(/red.jpg)');
        }
        else if(data.pish == "False"){
            $("#info").text("Safe site");
            $("#info").css("color", "green");
            $("#image").css('background-image', 'url(/green.jpg)');
        }
        else{
            alert("Oh ohh Unable to Predict");
```

```
                }
                $("#loading").removeClass("fa fa-spinner fa-spin");
        });
}

$(document).ready(function(){
        $("#button").click(function(){
                $("#loading").addClass("fa fa-spinner fa-spin");
                transfer();
        });
});
```

Popup.html

```
<!DOCTYPE html>
<html>
        <head>
                <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
                <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
                <script src="popup.js" type="text/javascript"></script>
                <link rel="stylesheet" href="style.css">
        </head>
        <body>
                <div class="card">
                        <div class="card_circle transition" id="image"></div>
                        <h2>Check for Phishing<br><small id = "info">Under
Development(Beta)</small></h2>
                        <div class="cta-container">
                                <a class="cta" id="button"> <i id = "loading"></i> Check !</a>
                        </div>
                </div>
```

```
        </body>
</html>
```

Style.css

```
@import
url(https://fonts.googleapis.com/css?family=Noto+Sans:400,700,400italic,700italic);
html, body {
  background-color: #eaeaea;
  height: 300px;
  width: 300px;
}
.card {
  background-color: #fff;
  position: absolute;
  margin: auto;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  overflow: hidden;
}
.card_circle {
  height: 400px;
  width: 450px;
  background: url(/Yellow.jpg) no-repeat;
  background-size: cover;
  position: absolute;
  border-radius: 50%;
  margin-left: -75px;
  margin-top: -270px;
}
h2 {
```

```
  text-align: center;
  margin-top: 150px;
  position: absolute;
  z-index: 9999;
  font-size: 26px;
  color: #2196F3;
  width: 100%;
  font-family: 'Noto Sans', serif;
}
h2 small {
  font-weight: normal;
  font-size: 65%;
  color: rgba(0,0,0,0.5);
}
.cta-container {
  text-align: center;
  margin-top: 250px;
  position: absolute;
  z-index: 9999;
  width: 100%;
  font-family: 'Noto Sans', serif;
}
.cta {
  color: #fff;
  border: 2px solid #2196F3;
  background-color: #2196F3;
  padding: 10px 25px;
  border-radius: 50px;
  font-size: 17px;
  text-decoration: none;
  font-weight: bold;
  cursor: pointer;
```

}

6.5.2 Server: Flask Server with acts as a central api server where Machine learning model is deployed and on api request it will validate the key and then extract the features of the received url and predict the result using the loaded Machine learning classifier and then send the response back to the client.

Source Code for Server:

App.py

```
from flask import Flask, request, jsonify, render_template
import joblib
import random
import test
import database
app = Flask(__name__)
app.config["DEBUG"] = True

apiKey = "vXSDrKoBvByAPkIzQIfQuiYDLx2OvH7wIypN67Mn"

model = joblib.load(ArtificialNeuralNetwork_9620.pkl')
print("Pickle Unpacked")

@app.route("/")
def index():
   return render_template('Home.html')

@app.route("/Dashboard")
def dashboard():
   return render_template('Dashboard/index.html')

@app.route("/api/v2/pishcheck",methods=['POST','GET'])
```

```
def api2():
    print("\n")
    print("#################### API V2 Called ####################")
    flag = False
    if request.method == 'POST':
        key = request.form['key']
        url = request.form['url']
        flag=True
    elif request.method == 'GET':
        if 'key' in request.args and 'url' in request.args:
            key = str(request.args['key'])
            url = str(request.args['url'])
            flag=True
        else:
            return "Please Follow the format ------> ?key=key&url=url"
    if(flag):
        if key == apiKey:
            print("Key Validation Successful")
            x = test.check(url,model)
            ret = {'url':url, 'pish':str(x)}
            return jsonify(ret)
        else:
            print("Key Validation Failed")
            return "Key Validation failed"
    else:
        return "Please Follow the format ------> ?key=key&url=url"

@app.errorhandler(404)
def page_not_found(e):
    return render_template("PageNotFound.html")
```

Test.py

```
import features_extraction
import sys
import joblib
import numpy as np
import time

def check(url,clf):
    fT = time.time()
    features_test = features_extraction.main(url)
    feT = time.time()
    features_test = np.array(features_test).reshape((1, -1))
    prT=time.time()
    pred = clf.predict(features_test)
    preT=time.time()
    print("\n")
    print("Feature extraction Time = "+str(feT-fT))
    print("Prediction Time = "+str(preT-prT))
    print("Total Time = "+str(preT-fT))
    print("\n")

    if int(pred[0][0])>0.51:
        print(url)
        print ("This is a safe website.")
        return False

    else:
        print(url)
        print ("This is a phishing website..!")
        return True
```

Feature_Extractor.py

```
# 1 stands for legitimate
# 0 stands for suspicious
# -1 stands for phishing

from bs4 import BeautifulSoup
import urllib
import bs4
import re
import socket
import whois
from datetime import datetime
import time
from urllib.request import Request, urlopen
from googlesearch import search
import sys

LOCALHOST_PATH = "/Library/WebServer/Documents/"

def having_ip_address(url):
    match = re.search('(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\."([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|"((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})\\/)"(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)
    if match:
        return -1
    else:
        return 1

def url_length(url):
    if len(url) < 54:
        return 1
```

```
    elif len(url) >= 54 | len(url) <= 75:
        return 0
    else:
        return -1

def shortening_service(url):
    match =
re.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|Bu
dURL\.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|bit
ly\.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|vzt
url\.com|qr\.net|1url\.com|tweez\.me|v\.gd|"tr\.im|link\.zip\.net', url)
    if match:
        return -1
    else:
        return 1
def having_at_symbol(url):
    match = re.search('@', url)
    if match:
        return -1
    else:
        return 1

def double_slash_redirecting(url):
    list = [x.start(0) for x in re.finditer('//', url)]
    if list[len(list) - 1] > 6:
        return -1
    else:
        return 1
```

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

def having_sub_domain(url):
    if having_ip_address(url) == -1:
        match = re.search('(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-
5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\."([01]?\\d\\d?|2[0-4]\\d|25[0-5]))|(?:[a-fA-F0-
9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)
        pos = match.end(0)
        url = url[pos:]
    list = [x.start(0) for x in re.finditer('\.', url)]
    if len(list) <= 3:
        return 1
    elif len(list) == 4:
        return 0
    else:
        return -1

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

    registration_length = 0
    if expiration_date:
        registration_length = abs((expiration_date - today).days)
    if registration_length / 365 <= 1:
        return -1
```

```
    else:
        return 1

def favicon(wiki, soup, domain):
    for head in soup.find_all('head'):
        for head.link in soup.find_all('link', href=True):
            dots = [x.start(0) for x in re.finditer('\.', head.link['href'])]
            if wiki in head.link['href'] or len(dots) == 1 or domain in head.link['href']:
                return 1
            else:
                return -1
    return 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

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

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

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

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

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

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


def url_of_anchor(wiki, soup, domain):
    i = 0
```

```
    unsafe = 0
    for a in soup.find_all('a', href=True):
        if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto" in a['href'].lower() or not (
                    wiki in a['href'] or domain in a['href']):
            unsafe = unsafe + 1
        i = i + 1
    try:
        percentage = unsafe / float(i) * 100
    except:
        return 1
    if percentage < 31.0:
        return 1
    elif 31.0 <= percentage < 67.0:
        return 0
    else:
        return -1


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

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

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

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

def sfh(wiki, soup, domain):
    for form in soup.find_all('form', action=True):
        if form['action'] == "" or form['action'] == "about:blank":
            return -1
        elif wiki not in form['action'] and domain not in form['action']:
            return 0
        else:
            return 1
    return 1

def submitting_to_email(soup):
    for form in soup.find_all('form', action=True):
        if "mailto:" in form['action']:
            return -1
        else:
            return 1
    return 1

def abnormal_url(domain, url):
    hostname = domain.name
    match = re.search(hostname, url)
```

```
    if match:
        return 1
    else:
        return -1

def i_frame(soup):
    for i_frame in soup.find_all('i_frame', width=True, height=True, frameBorder=True):
        if i_frame['width'] == "0" and i_frame['height'] == "0" and i_frame['frameBorder'] ==
"0":
            return -1
        else:
            return 1
    return 1

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

def web_traffic(url):
    try:
        rank =
bs4.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

def google_index(url):
    site = search(url, 5)
    if site:
        return 1
    else:
        return -1
def statistical_report(url, hostname):
    url_match = re.search(
'at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|sweddy\.com|myjino\.ru|96\.lt|o
w\.ly', url)
    try:
        ip_address = socket.gethostbyname(hostname)
    except:
        print('Connection problem. Please check your internet connection!')
    ip_match =
re.search('146\.112\.61\.108|213\.174\.157\.151|121\.50\.168\.88|192\.185\.217\.116|78\.46\.2
11\.158|181\.174\.165\.13|46\.242\.145\.103|121\.50\.168\.40|83\.125\.22\.219|46\.242\.145\.9
8|"107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.151\.148\.108|
107\.151\.148\.109|119\.28\.52\.61|54\.83\.43\.69|52\.69\.166\.231|216\.58\.192\.225|"118\.18
4\.25\.86|67\.208\.74\.71|23\.253\.126\.58|104\.239\.157\.210|175\.126\.123\.219|141\.8\.224\.
221|10\.10\.10\.10|43\.229\.108\.32|103\.232\.215\.140|69\.172\.201\.153|"216\.218\.185\.162|
54\.225\.104\.146|103\.243\.24\.98|199\.59\.243\.120|31\.170\.160\.61|213\.19\.128\.77|62\.11
3\.226\.131|208\.100\.26\.234|195\.16\.127\.102|195\.16\.127\.157|"34\.196\.13\.28|103\.224\.
212\.222|172\.217\.4\.225|54\.72\.9\.51|192\.64\.147\.141|198\.200\.56\.183|23\.253\.164\.103
|52\.48\.191\.26|52\.214\.197\.72|87\.98\.255\.18|209\.99\.17\.27|"216\.38\.62\.18|104\.130\.12
```

```
4\.96|47\.89\.58\.141|78\.46\.211\.158|54\.86\.225\.156|54\.82\.156\.19|37\.157\.192\.102|204\
.11\.56\.48|110\.34\.231\.42',
        ip_address)
    if url_match:
        return -1
    elif ip_match:
        return -1
    else:
        return 1
def main(url):
    soup_string = urlopen(Request(url, headers={'User-Agent': 'Mozilla/5.0'}))
    soup = BeautifulSoup(soup_string, 'html.parser')
    status = []
    hostname = url
    h = [(x.start(0), x.end(0)) for x in
re.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 re.finditer('/', hostname)]
        z = int(len(h))
        if z != 0:
            hostname = hostname[:h[0][0]]
    status.append(having_ip_address(url))
    status.append(url_length(url))
    status.append(shortening_service(url))
    status.append(having_at_symbol(url))
    status.append(double_slash_redirecting(url))
    status.append(prefix_suffix(hostname))
    status.append(having_sub_domain(url))
    dns = 1
```

```
try:
    domain = whois.query(hostname)
except:
    dns = -1
if dns == -1:
    status.append(-1)
else:
    status.append(domain_registration_length(domain))
status.append(favicon(url, soup, hostname))
status.append(https_token(url))
status.append(request_url(url, soup, hostname))
status.append(url_of_anchor(url, soup, hostname))
status.append(links_in_tags(url, soup, hostname))
status.append(sfh(url, soup, hostname))
status.append(submitting_to_email(soup))
if dns == -1:
    status.append(-1)
else:
    status.append(abnormal_url(domain, url))
status.append(i_frame(soup))
if dns == -1:
    status.append(-1)
else:
    status.append(age_of_domain(domain))
status.append(dns)
status.append(web_traffic(soup))
status.append(google_index(url))
status.append(statistical_report(url, hostname))
print(status)
return status
```

6.5.3 Machine Learning Model: We have tested classification accuracy using different ML models and due to the reduced size of the model object and high accuracy, we have selected Sequential Artificial network with five hidden layer as the predictor for our application.

Sequential Artificial Neural Network Model Creation and Exporting

```
import numpy as np
from sklearn.model_selection import train_test_split
from keras import Sequential
from keras.layers import Dense

#Data Processing
labels=[]
features=[]
file=open('Training Dataset.arff').read()
list=file.split('\n')
data=np.array(list)
data_new=[i.split(',') for i in data]
data_new=data_new[0:-1]
for i in data_new:
   labels.append(i[30])
data_new = np.array(data_new)
features = data_new[:,:-1]
features = features[:, [0, 1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17, 22, 23, 24, 25, 27, 29]]
features = np.array(features).astype(np.int)
labels = np.array(labels).astype(np.int)
features_train, features_test, labels_train, labels_test = train_test_split(features, labels, test_size=0.33, random_state=42)

#Artificial Neural Network
classifier = Sequential()
#First Hidden Layer
classifier.add(Dense(22,activation='relu',kernel_initializer='RandomUniform',input_dim=22))
```

```
#Second  Hidden Layer
classifier.add(Dense(30, activation='relu', kernel_initializer='random_normal'))

#Third  Hidden Layer
classifier.add(Dense(18, activation='relu', kernel_initializer='TruncatedNormal'))

#Fourth  Hidden Layer
classifier.add(Dense(16, activation='relu', kernel_initializer='random_normal'))

#Fifth  Hidden Layer
classifier.add(Dense(6, activation='relu', kernel_initializer='random_normal'))

#Output Layer
classifier.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))

#Compiling the neural network
classifier.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])

#Fitting the data to the training dataset
classifier.fit(features_train,labels_train, batch_size=100, epochs=5000, verbose = True)

eval_model=classifier.evaluate(features_train, labels_train)
eval_model

#Predicting
y_pred=classifier.predict(features_test)
y_pred =[1 if pred>0.5 else -1 for pred in y_pred]

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
```

```
#Accuracy report
print(classification_report(labels_test, y_pred))
print('The accuracy is:', 100*accuracy_score(labels_test, y_pred),'%')
print(confusion_matrix(labels_test, y_pred))

#Exporting model to pickle file
import joblib
joblib.dump(classifier, 'ArtificialNeuralNetwork.pkl')
```

Importing Model and Testing

```
import joblib
import numpy as np
classifier = joblib.load('ArtificialNeuralNetwork.pkl')
import features_extraction

# Test for https://www.google.com (Safe Page)
feature = features_extraction.main("https://www.google.com/")
features_test = np.array(feature).reshape((1, -1))
ans = classifier.predict(features_test)
print(ans[0][0]<0.5) #True

# Test for https://www.llbonue.vot.pl (Deceptive Page)
feature = features_extraction.main("https://www.llbonue.vot.pl/")
features_test = np.array(feature).reshape((1, -1))
print(features_test)
ans = classifier.predict(features_test)
print(ans[0][0]<0.5) #False
```

CHAPTER 7

TESTING

We have tested the modules used in our system:

7.1. ML Model Testing

- Accuracy Test
- Memory Usage Test

7.2. API Testing

- Unit Testing: While the testing world can be filled with misnomers, the easiest way to think about a "unit test" and APIs is testing a single endpoint, with a single request, looking for a single response or set of responses.
- Integration Testing: Integration testing is the most often used form of API testing, as APIs are at the centre of most integrations between internal or third-party services
- End-to-End Testing: End-to-End testing can help us validate the flow of data and information between a few different API connections.
- Performance Testing: The
- Security testing: Our API is not accessible without a secret api key

7.3. Chrome Extension Testing

- Browser compatibility testing
- User interface testing
- Functionality testing
- Compatibility testing & configuration testing

CHAPTER 8

RESULTS

Results obtained by ANN show that ANN has higher achievement compared to other classifier (SVM, KNN, RF and LGBM) methods. This study is considered to be an applicable design in automated systems with high performing classification against the phishing activity of websites.

When the user enters a URL, the extension takes the URL using the POST method and passes the same to the python code at the server through API using the Java script of the extension. The python code then extracts all the features from the URL and forms an array of features. It then predicts the result using pre-loaded trained Artificial Neural Network.

8.1. Snapshots

Below is the screenshot of a safe website i.e. http://www.stackoverflow.com. Since major of the features of the website were safe, the net result was obtained as safe by the Chrome extension. We display the complete web and future scope site to the user.

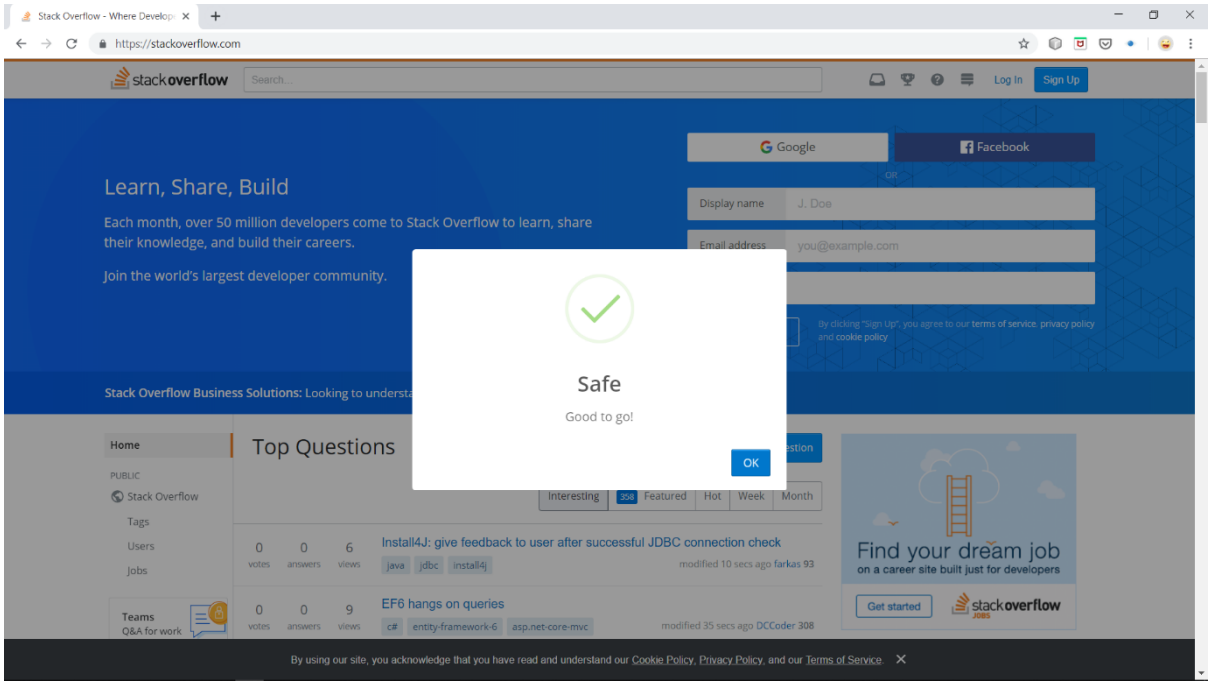

fig 8.1.1 Result of the extension for a safe website

Now, for the case of phishing websites, if the website is rendered as phishing by the extension, the user is prompted about it. Below is the screenshot for the deceptive websites i.e http://secureyourpayment.fr/.

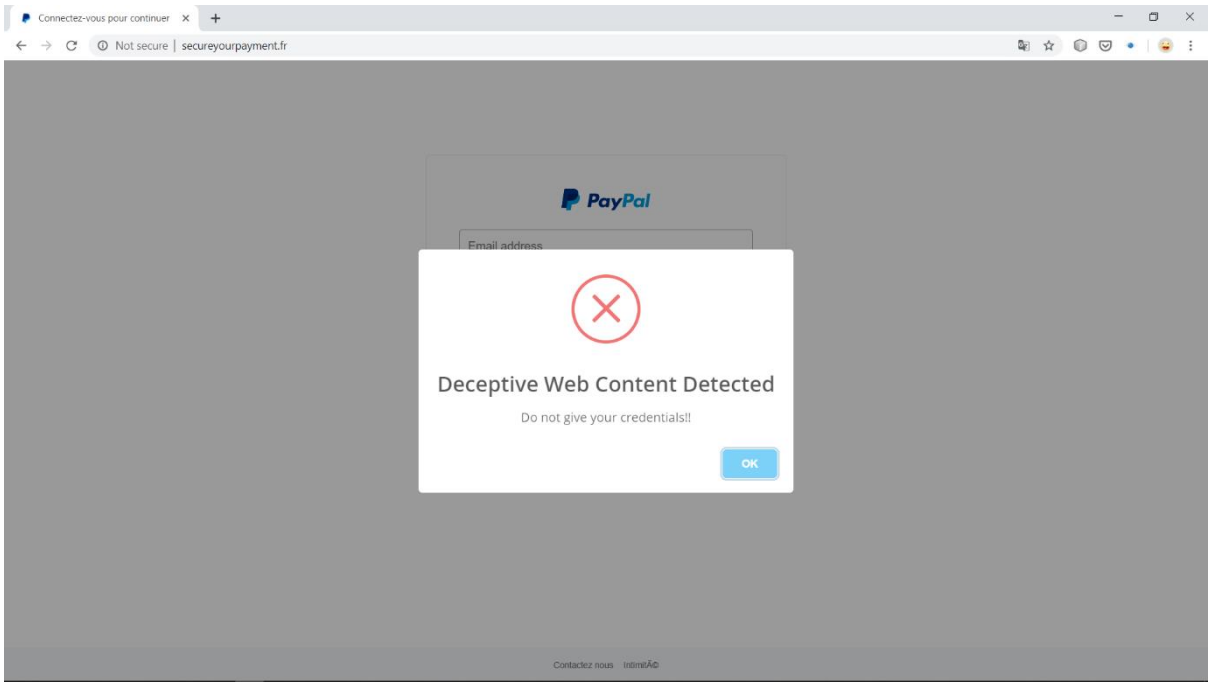

fig 8.1.2 Result of the extension for a Deceptive website

CHAPTER 9

FUTURE ENHANCEMENTS

The future scope of this idea is very broad. Some websites only have a few components (for example, a form, or an image etc.) which make the website unsafe. So, for such websites, one can block this malicious part and display the complete safe webpage to the users. This can be implemented in a similar fashion as to that of AdBlock Extension which blocks a particular part of the webpage and displays the rest.

So, for such AdBlock sites our extension will help in blocking all the types of ads being displayed where they actually lead to their sites instead of the one which we are looking for which is very useful feature in the websites. It also focuses on the safety gateways of the banking procedures by making the best use of the encryption methods and checking whether all the features are matching to our feature extractor to make sure that the site is safe.

The Extension and API Server can be enhanced to detect other different attacks carried out in Cyber world. Hence providing a safe and secure browsing experience for the end user.

Currently the Sequential Artificial Network is a Static trained model. it can be Trained Dynamically That is, data is continually entering the system and incorporating that data into the model through continuous updates. Hence increasing the accuracy of the model with time by training it dynamically.

CHAPTER 10

CONCLUSION

This Project proposes the development of a Chrome Extension for identifying deceptive websites. This concept displays the safety component of a website to keep the user safe. Otherwise, the user might end up giving his credentials to the phishers which can lead to huge losses.

Since, Google Chrome being a widely used web browser, Development of Chrome extension for identifying deceptive websites is a safety component to keep the user safe from these attacks so that the user doesn't end up giving his credentials to the phishers which can lead to huge losses. The purpose of the application is to make a classification of the website visited by the user by extracting its features for the determination of one of the types of attacks that cyber threats called phishing. Sequential Artificial Neural Network is used for this purpose. In this, we used a data set from UCI Machine Learning Repository.

This API Server consists of feature extraction from the received URL and classification of features as safe or Deceptive using loaded Pre-Trained Sequential Artificial Neural Network. In the feature extraction, we have clearly defined rules of phishing feature extraction and these rules have been used for obtaining features of url received from API request.