VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi, Karnataka-590018

A MINI PROJECT REPORT

ON

PHISHING DETECTION PLUGIN

Submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & BUSINESS SYSTEM

Submitted by

4CB21CB019	Himanshu
4CB21CB052	Shravya R
4CB21CB050	Sathwik
4CB21CB060	Varun Shetty

Under the guidance of

Mrs. Ashwini K.G

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND BUSINESS SYSTEM

CANARA ENGINEERING COLLEGE

SUDHINDRA NAGARA, BENJANAPADAVU, BANTWAL – 574219

2023-2024

CANARA ENGINEERING COLLEGE

BANTWAL, D.K. 574219-KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE AND BUSINESS SYSTEM

CERTIFICATE

This is to certify that the mini project work entitled **PHISHING DETECTION PLUGIN** carried out by **Mr. Himanshu (4CB21CB019)**, **Ms. Shravya R (4CB21CB052)**, **Mr.Sathwik (4CB21CB050)** and **Mr.Varun Shetty (4CB21CB060)**, bonafide students of **CANARA ENGINEERING COLLEGE, BENJANAPADAVU** in partial fulfillment for the award of **BACHELOR OF ENGINEERING** in **COMPUTER SCIENCE AND BUSINESS SYSTEM** of the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during the year **2023 - 2024**. The project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the said Degree.

Signature of Guide
Prof. Ashwini K.G

Signature of HOD
Dr. Udaya Kumar K Shenoy

ABSTRACT

This document proposes a phishing detection plugin for chrome browser that can detect and warn the user about phishing web sites in real-time using random forest classifier. Based on the IEEE paper, Intelligent phishing website detection using random forest classifier, the random forest classifier seems to outperform other techniques in detecting phishing websites. One common approach is to make the classification in a server and then let the plugin to request the server for result. Unlike the old approach, this project aims to run the classification in the browser itself. The advantage of classifying in the client side browser is, better privacy (the user's browsing data need not leave his machine), detection is independent of network latency. This project is mainly of implementing the above mentioned paper in JavaScript for it to run as a browser plugin. Since JavaScript doesn't have much ML libraries support and considering the processing power of the client machines, the approach needs to be made lightweight. The random forest classifier needs to be trained on the phishing websites dataset using python scikit-learn and then the learned model parameters need to be exported into a portable format for using in JavaScript.

ACKNOWLEDGEMENT

It is our great pleasure to acknowledge the assistance and contributions of all the people who helped us to make our Mini Project successful.

We wish to express our deepest gratitude to project guide **Prof. Ashwini K.G, Department of Computer Science and Business System, CEC, Mangalore**, for her invaluable support, guidance, and encouragement throughout the duration of this project.

We extend our thanks to **Prof. Pooja Kini, Project Coordinator, Department of Computer Science and Business System, CEC, Mangalore**, for her exceptional support and coordination throughout this project.

We are extremely grateful to **Dr. Udaya Kumar K Shenoy, Head of the Computer Science and Business System Department, CEC, Mangalore**, for his moral support and valuable suggestions throughout this project.

We would like to express our deepest gratitude to our **Principal, Dr. Nagesh H R** whose unwavering support and guidance have been instrumental in the successful completion of this project.

We thank all the faculty and technical staff of Department of Computer Science and Business System for their kind help

Himanshu (4CB21CB019)
Shravya R (4CB21CB052)
Sathwik (4CB21CB050)
Varun Shetty (4CB21CB060)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF Tables

CHAPTER 1

Introduction

1.1 Overview

In the digital age, the prevalence of phishing attacks has become a significant threat to online security. Phishing is a form of cybercrime that involves tricking users into providing sensitive information, such as usernames, passwords, and financial details, by masquerading as a trustworthy entity in electronic communications. These attacks can occur through various channels, including email, social media, and websites, often employing sophisticated tactics to deceive users. As techniques used by cybercriminals become more advanced, there is an urgent need for effective tools to help users identify and avoid phishing websites. The impact of phishing attacks can be severe, leading to financial losses, identity theft, and unauthorized access to personal and corporate information. Traditional methods of combating phishing, such as blacklisting known phishing sites, often fall short due to the dynamic nature of these attacks. Therefore, leveraging machine learning techniques to develop more robust and adaptive phishing detection systems is becoming increasingly important.

This project presents a Chrome extension designed to detect phishing websites while ensuring user privacy. By implementing a client-side classification model, the extension analyzes URLs in real-time and provides immediate alerts to users when they attempt to access potentially harmful sites. The core of this solution is a Random Forest Classifier, trained on a public dataset from the UCI Repository, which allows the extension to classify URLs based on their characteristics without collecting any user browsing data. This approach not only enhances security but also respects user privacy, addressing a common concern in the development of security tools.

The primary objective of this project is to enhance online security by empowering users with a reliable tool for phishing detection. By prioritizing privacy and user-friendly design, the extension aims to facilitate safer browsing experiences for individuals, enabling them to navigate the internet with confidence.

1.2 Problem Statement

Phishing attacks are increasingly sophisticated and frequent, posing a major threat to online security by tricking users into revealing sensitive information. Traditional methods like blacklisting known phishing sites are ineffective due to the dynamic nature of these attacks and often compromise user privacy by collecting browsing data. There is a critical need for a privacy-preserving tool that can accurately detect phishing websites in real-time.

1.3 Scope of the project

According to Wikipedia, In 2017, 76% of organisations experienced phishing attacks. Nearly half of information security professionals surveyed said that the rate of attacks increased from 2016. In the first half of 2017 businesses and residents of Qatar were hit with more than 93,570 phishing events in a three-month span. With increasing number of internet users, there is a prominent need for security solutions again attacks such as phishing. Hence this plugin would be a good contribution for the chrome users.

1.4 Objective

The objective of the project is to develop a browser plugin that can accurately identify and warn users about phishing websites. This plugin leverages a machine learning model, specifically a Random Forest Classifier, trained to recognize patterns and features typical of phishing websites. The ultimate goal is to enhance user safety by reducing the risk of phishing attacks, providing users with real-time alerts and relevant information about the websites they visit.

CHAPTER 2

Literature Survey

An existing chrome plugin named PhishDetector uses a rule based approach so that it can detect phishing without external web service. Although rule based approaches support easier implementation on client side, they can't be accurate compared to Machine Learning based approaches. Similar work by Shreeram.V on detection of phishing attacks using genetic algorithm6 algorithm for detection.

Phishwish: A Stateless Phishing Filter Using Minimal Rules by Debra L. Cook, Vijay K. Gurbani, Michael Daniluk worked on a phishing filter that offers advantages over existing filters: It does not need any training and does not consult centralized white or black lists. They used only 11 rules to determine the veracity of an website.

Intelligent phishing website detection using random forest classifier (IEEE-2017) by Abdulhamit Subasi, Esraa Molah, Fatin Almkallawi and Touseef J. Chaudhery discusses the use the random forest classifier for phishing detection. Random Forest has performed the best among the classification methods by achieving the highest accuracy 97.36

PhishBox: An Approach for Phishing Validation and Detection (IEEE-2017) by Jhen-Hao Li, and Sheng-De Wang discusses ensemble models for phishing detection. As a result, The false-positive rate of phishing detection is dropped by 43.7% in average.

Real time detection of phishing websites (IEEE-2016) by Abdul- ghani Ali Ahmed, and Nurul Amirah Abdullah discusses an approach based on features from only the URL of the website. They were able to come up with a detection mechanism that is capable of detecting various types of phishing attacks maintaining a low rate of false alarms.

Using Domain Top-page Similarity Feature in Machine Learning- Based Web Phishing Detection by Nuttapong Sanglerdsinlapachai,Arnon Rungsawang presents a study on using a concept feature to detect web phishing problem. They applied additional domain top-page similarity feature to a machine learning based phishing detection system. The evaluation result in terms of f-measure was up to 0.9250, with 7.50% of error rate.

CHAPTER 3

System Requirement Specification

3.1 Functional Requirements

The plugin warns the user when he/she visits a phishing website. The plugin should adhere to the following requirements:

- The plugin should be fast enough to prevent the user from submitting any sensitive information to the phishing website.
- The plugin should not use any external web service or API which can leak user's browsing pattern.
- The plugin should be able to detect newly created phishing websites.
- The plugin should have a mechanism of updating itself to emerging phishing techniques.

3.2 Non-Functional Requirements

3.2.1 User Interface

There must be a simple and easy to use user interface where the user should be able to quickly identify the phishing website. The input should be automatically taken from the webpage in the current tab and the output should be clearly identifiable. Further, the user should be interrupted on the event of phishing.

3.2.2 Software

- Python for training the model
- Chrome browser

3.2.3 Performance

The plugin should be always available and should make fast detection with low false negatives.

3.3 Constraints and Assumptions

3.3.1 Constraints

- Certain techniques use features such as SSL page rank etc. Such information cannot be obtained from client side plugin without external API. Thus those features can't be used for prediction.
- Heavy techniques can't be used considering the processing power of client machines and the page load time of the website.
- Only JavaScript can be used to develop Chrome plugins. Machine learning libraries support for JavaScript is far less compared to Python and R.

3.3.2 Assumptions

- The plugin is provided with the needed permissions in the Chrome environment.
- The user has a basic knowledge about phishing and extensions.

3.4 System Models

3.4.1 Use Case Diagram

The overall use case diagram of the entire system is shown in Figure 1. The user can install the plugin and then can continue his normal browsing behaviour. This plugin will automatically check the browsing pages for phishing and warns the user of the same.

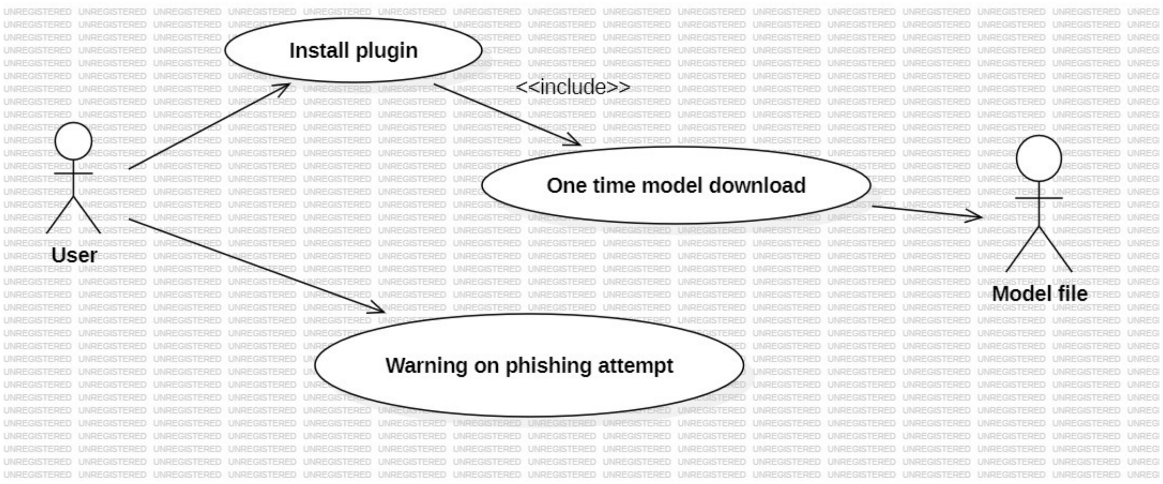


Figure 1: Use case diagram of the system

3.4.2 Sequence Diagram

The sequence of interactions between the user and the plugin are shown in Figure 2.

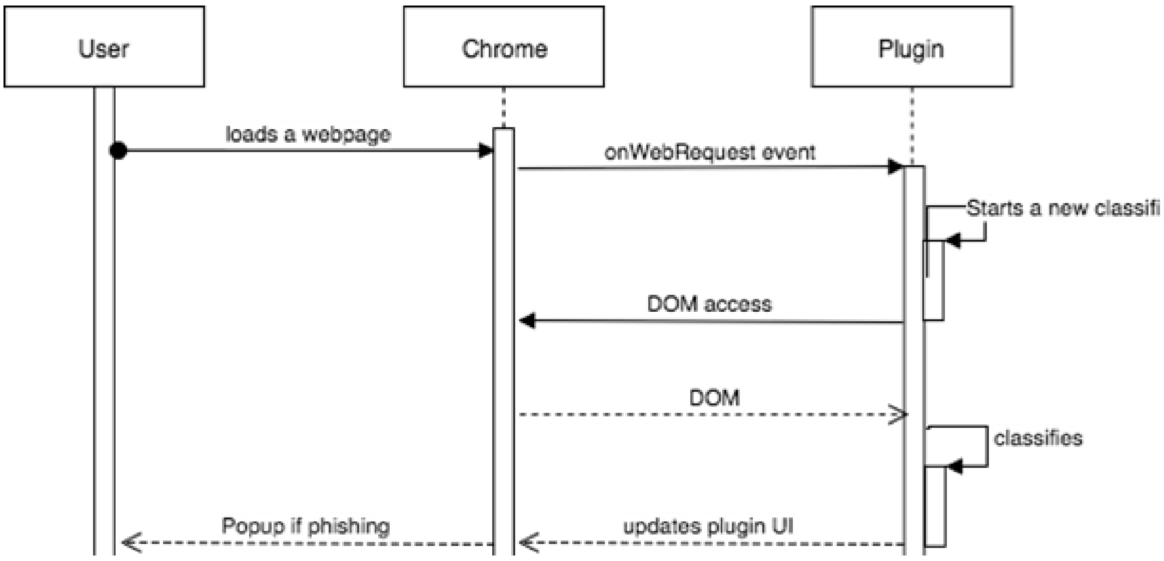


Figure 2: System Sequence diagram

CHAPTER 4

System Design

4.1 Architectural Design

The block diagram of the entire system is shown in Figure 3. A Random Forest classifier is trained on a phishing sites dataset using Python's scikit-learn. A JSON format to represent the Random Forest classifier has been devised and the learned classifier is exported to the same. A browser script has been implemented which uses the exported model JSON to classify the website being loaded in the active browser tab.

The dataset arff file is loaded using the Python arff library, and 17 features are chosen from the existing 30 features. Features are selected based on the ability to be extracted completely offline on the client side without being dependent on a web service or third party. The dataset with chosen features is then separated for training and testing. The Random Forest is trained on the training data and exported to the aforementioned JSON format. The JSON file is hosted on a URL.

The client-side Chrome plugin executes a script on each page load, extracting and encoding the selected features. Once the features are encoded, the plugin checks for the exported model JSON in the cache and downloads it again if it is not present.

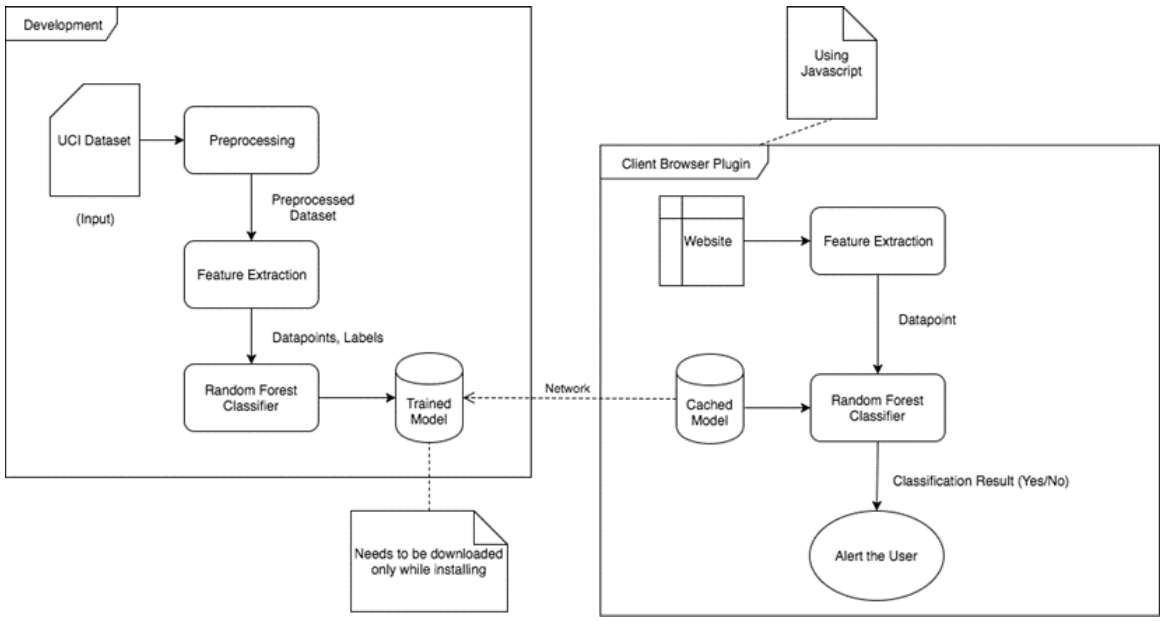


Figure 3: System Architecture

With the encoded feature vector and model JSON, the script can run the classification. A warning is displayed to the user if the website is classified as phishing. The entire system is designed to be lightweight, ensuring rapid detection.

4.2 UI Design

A simple and easy-to-use User Interface (UI) has been designed for the plugin using HTML and CSS. The UI contains a large circle indicating the percentage of the legitimacy of the website in the active tab. The circle changes color with respect to the classification output (Green for legitimate website and Light Red for phishing). Below the circle, the analysis results containing the extracted features are displayed in the following color code:

- Green - Legitimate
- Yellow - Suspicious
- Light Red - Phishing

The plugin also displays an alert warning in case of phishing to prevent the user from entering any sensitive information on the website. The test results such as precision, recall, and accuracy are displayed on a separate screen. The UI is shown in Figure 4.

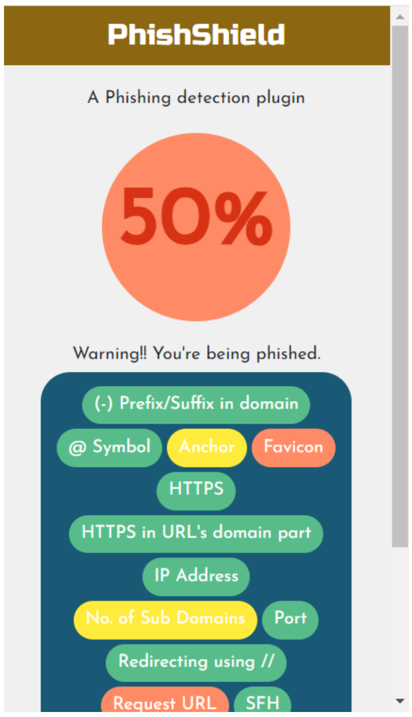

PhishShield
A Phishing detection plugin
50%
Warning!! You're being phished.
(-) Prefix/Suffix in domain
@ Symbol
Anchor
Favicon
HTTPS
HTTPS in URL's domain part
IP Address
No. of Sub Domains
Port
Redirecting using //
Request URL
SFH

Figure 4: UI Design

4.3 Class Diagram

The class diagram of the entire system is shown in Figure 5. This diagram depicts the functions of various modules in the system clearly. It also shows the interaction between the modules, thereby providing a clear idea for implementation.

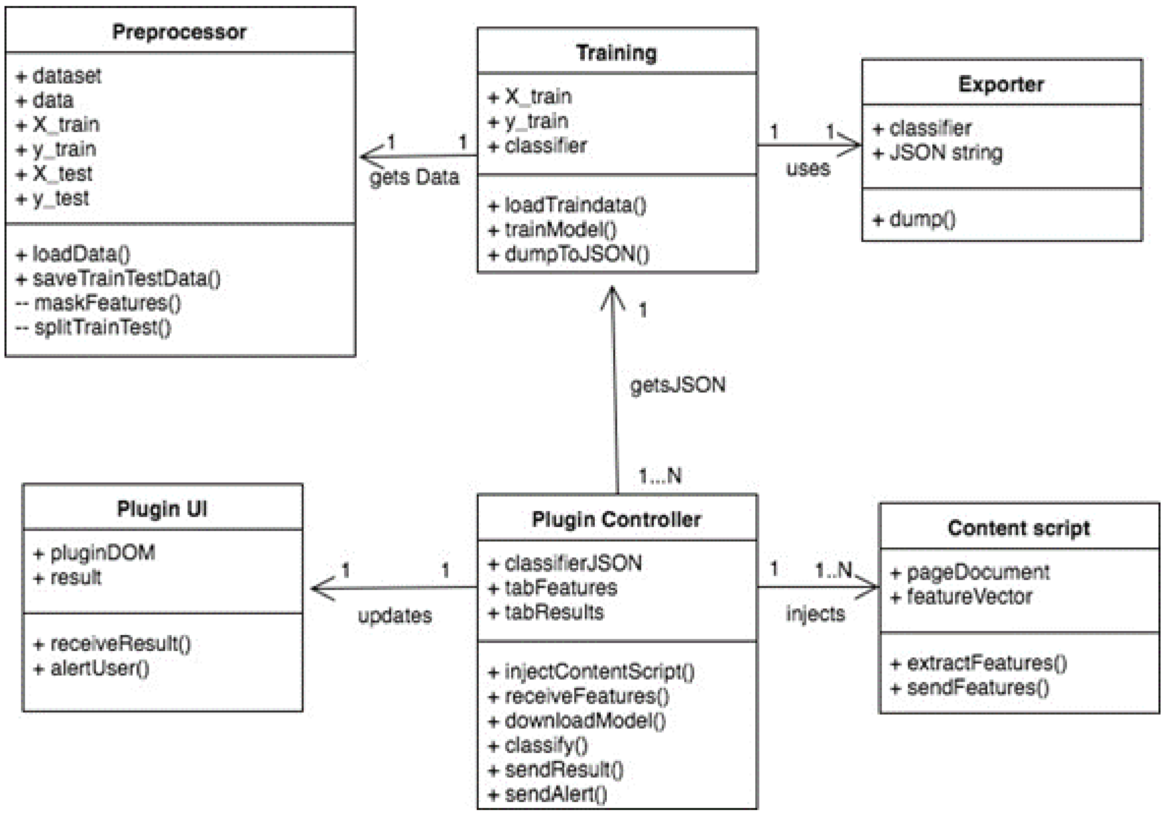


Figure 5: Class Diagram

4.4 Module Design

4.4.1 Preprocessing

The dataset is downloaded from the UCI repository and loaded into a NumPy array. The dataset consists of 30 features that need to be reduced for extraction on the browser. Each feature is experimented with on the browser to ensure it can be feasibly extracted without using any external web service or third party. Based on the experiments, 17 features have been chosen out of 30 with minimal accuracy loss on the test data. More features increase accuracy but reduce the ability to detect rapidly, considering the feature extraction time. Thus, a subset of features is chosen to balance this tradeoff.

Feature	**Description**
IP address	
URL length	
URL shortener	
'@' in URL	
Redirection with '//'	
'-' in domain	
Degree of subdomain	
HTTPS	
Favicon domain	
TCP Port	
HTTPS in domain name	
Anchor tag href domains	
Script & link tag domains	
Empty server form handler	
Use of mailto	
Use of iFrame	
Cross domain requests	

Table 1: Webpage Features

The dataset is split into training and testing sets with 30% for testing. Both the training and testing data are saved to disk.

4.4.2 Training

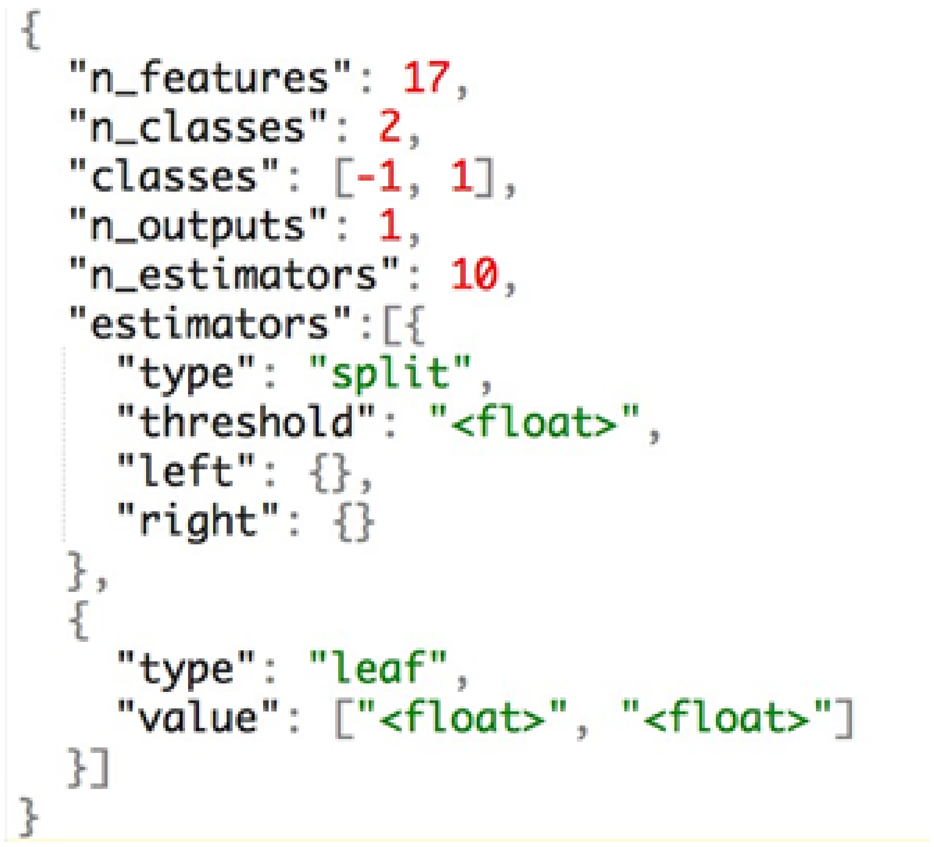
The training data from the preprocessing module is loaded from the disk. A Random Forest classifier is trained on the data using the scikit-learn library. Random Forest is an ensemble learning technique using an ensemble of 10 decision tree estimators. Each decision tree follows the CART algorithm and tries to reduce the Gini impurity.

$$\text{Gini}(E_j) = 1 - \sum_{j=1}^{c} p_j^2 \tag{1}$$

The cross-validation score is calculated on the training data, and the F1 score is calculated on the testing data. The trained model is then exported to JSON using the next module.

4.4.3 Exporting Model

Machine learning algorithms learn parameter values during the training phase. In Random Forest, each decision tree is an independent learner, learning node threshold values, and the leaf nodes learn class probabilities. A format needs to be devised to represent the Random Forest in JSON. The overall JSON structure consists of keys such as the number of estimators, number of classes, etc. Further, it contains an array in which each value is an estimator represented in JSON. Each decision tree is encoded as a JSON tree with nested objects containing the threshold for that node and left and right node objects recursively.



```
{
  "n_features": 17,
  "n_classes": 2,
  "classes": [-1, 1],
  "n_outputs": 1,
  "n_estimators": 10,
  "estimators":[{
    "type": "split",
    "threshold": "<float>",
    "left": {},
    "right": {}
  },
  {
    "type": "leaf",
    "value": ["<float>", "<float>"]
  }]
}
```

Figure 6: Random Forest JSON structure

4.4.4 Plugin Feature Extraction

This module takes a web page as input and generates a feature vector with 17 encoded features.

4.4.5 Classification

This module takes the feature vector from the feature extraction module and the JSON format from the exporting model module, then gives a boolean output denoting whether the web page is legitimate or phishing.

4.5 Complexity Analysis

4.5.1 Time Complexity

The time complexity of each module of the system is shown in Table 2.

S.No	**Module**	**Complexity**
1	Preprocessing	$O(n)$
2	Training	$O(E \cdot v \cdot n \log n)$
3	Exporting model	$O(E \cdot n \log n)$
4	Plugin feature extraction	$O(v)$
5	Classification	$O(E \cdot n \log n)$

Table 2: Time Complexity of Various Modules

- n denotes the number of data points.
- E denotes the number of ensembles (decision trees).
- v denotes the number of features.

4.5.2 Complexity of the Project

The complexity of the project lies in balancing the tradeoff between accuracy and rapid detection. Choosing a subset of features that enable fast detection without a significant drop in accuracy is crucial.

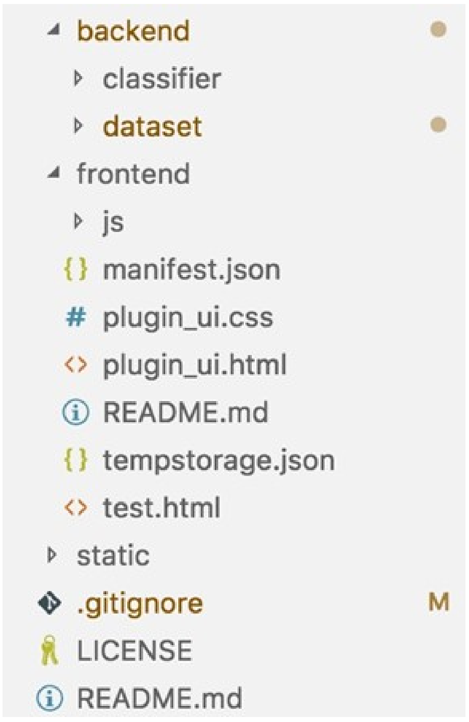
- Porting of scikit-learn Python object to JavaScript-compatible format, e.g., JSON.
- Reproducing the Random Forest behavior in JavaScript reduced the accuracy by a small margin.
- Many features are not feasible to extract without using an external web service. Using an external web service affects the detection time.
- Maintaining rapid detection is important as the system should detect phishing before the user submits any sensitive information.

CHAPTER 5

Implementation

5.1 System Development

The system is overall split into backend and plugin. The backend consists of dataset preprocessing and training modules. The frontend which is the plugin consists of JavaScript files for content script and background script including the Random Forest script. The plugin also consists of HTML and CSS files for the user interface. The overall code overview showing the organisation of these various modules can be seen in figure 7.



```
▲ backend                 ●
    ▷ classifier
    ▷ dataset             ●
▲ frontend
    ▷ js
    {} manifest.json
    # plugin_ui.css
    <> plugin_ui.html
    ⓘ README.md
    {} tempstorage.json
    <> test.html
▷ static
.gitignore                M
LICENSE
ⓘ README.md
```

Figure 7: Code Overview

5.2 Prototype Across the Models

The input and output to each module of the system is described in this section.

- Preprocessing: This module takes the downloaded dataset in arff format and the creates four new files listed as training features, training class labels, testing features, testing class labels.

- Training: This module takes the four output files from preprocessor and gives a trained Random Forest object along with the cross validation score on the training set.
- Exporting model: This module takes the learned Random Forest classifier object and the recursively generates its JSON represen- tation which is written to file in disk.
- Plugin Feature Extraction: This module takes a web page as in- put and gen-erates a feature vector with 17 encoded features.
- Classification: This module takes the feature vector from feature extraction module and the JSON format from the Exporting mod- el module and then gives a boolean output which denotes whether the web page is legitimate or phishing.

5.3 Exporting Algorithm

The algorithm used to export Random Forest model as JSON is as follows:

Tree_To_JSON(NODE):

```
tree_json = {}
if (node has threshold) then
    tree_json["type"] = "split"
    tree_json["threshold"] = node.threshold
    tree_json["left"] = TREE_TO_JSON(node.left)
    tree_json["right"] = TREE_TO_JSON(node.right)
else
    tree_json["type"] = "leaf"
    tree_json["values"] = node.values
return tree_json
```

Random_Forest_To_JSON(RF):

```
forest_json = {}
forest_json['n_features'] = rf.n_features_
forest_json['n_classes'] = rf.n_classes_
forest_json['classes'] = rf.classes_
forest_json['n_outputs'] = rf.n_outputs_
forest_json['n_estimators'] = rf.n_estimators
forest_json['estimators'] = []
e = rf.estimators
for (i = 0 to rf.n_estimators)
    forest_json['estimators'][i] = TREE_TO_JSON(e[i])
return forest_json
```

5.4 Deployment Details

The backend requires Python 3 and the Classifier JSON and Test set are served over HTTP using Github. The plugin is distributed as single file and requires Chrome browser to run. The plugin (frontend) is packed into a crx file for distribution.

CHAPTER 6

Results and Discussion

6.1 Dataset for Testing

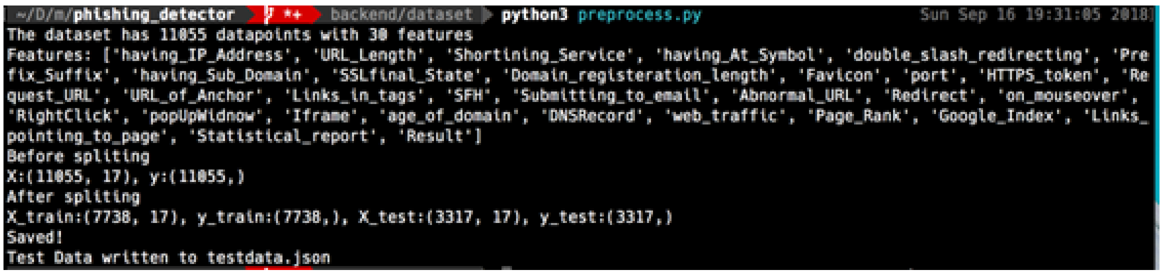
The test set consists of data points separated from the dataset by ratio 70:30. Also the plugin is tested with websites that are listed in PhishTank. New phishing sites are also added to PhishTank as soon as they are found. It should be noted that the plugin is able detect new phishing sites too. The results of this module testing as well as the testing of the entire system are summarised below.

6.2 Output Obtained in Various Stages

This section shows the results obtained during module testing.

6.2.1 Preprocessing

The output the preprocessing module is shown in figure 8.

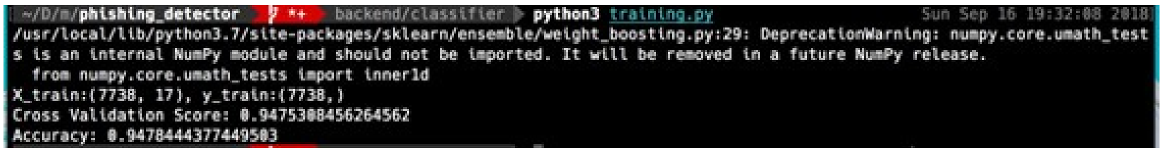


```
~/D/m/phishing_detector  *+  backend/dataset  python3 preprocess.py                Sun Sep 16 19:31:05 2018
The dataset has 11055 datapoints with 30 features
Features: ['having_IP_Address', 'URL_Length', 'Shortining_Service', 'having_At_Symbol', 'double_slash_redirecting', 'Pre
fix_Suffix', 'having_Sub_Domain', 'SSLfinal_State', 'Domain_registeration_length', 'Favicon', 'port', 'HTTPS_token', 'Re
quest_URL', 'URL_of_Anchor', 'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL', 'Redirect', 'on_mouseover',
'RightClick', 'popUpWidnow', 'Iframe', 'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank', 'Google_Index', 'Links_
pointing_to_page', 'Statistical_report', 'Result']
Before spliting
X:(11055, 17), y:(11055,)
After spliting
X_train:(7738, 17), y_train:(7738,), X_test:(3317, 17), y_test:(3317,)
Saved!
Test Data written to testdata.json
```

Figure 8: Preprocessing Output

6.2.2 Training

The output the training module is shown in figure 9.

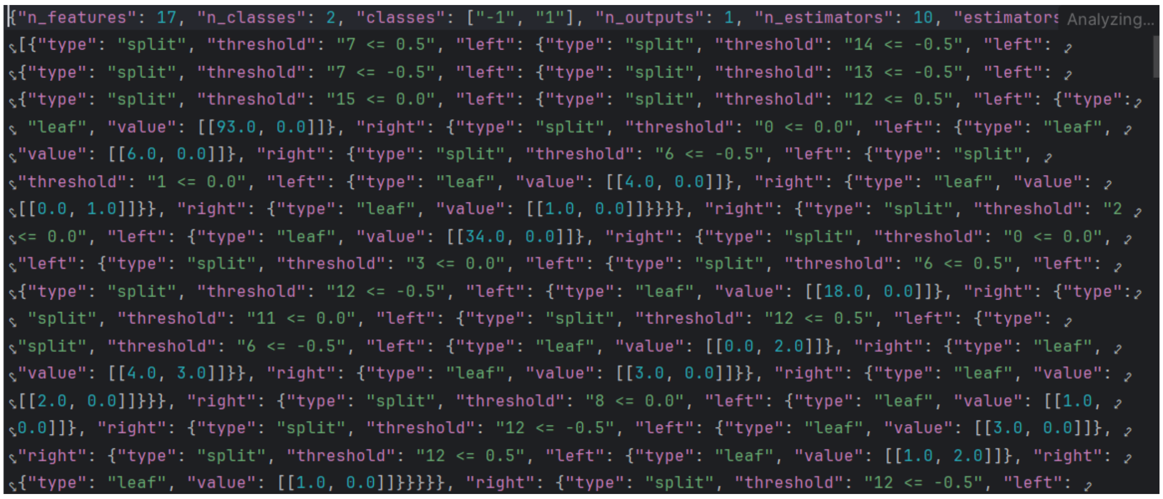


```
~/D/m/phishing_detector  *+  backend/classifier  python3 training.py               Sun Sep 16 19:32:08 2018
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_test
s is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
X_train:(7738, 17), y_train:(7738,)
Cross Validation Score: 0.9475308456264562
Accuracy: 0.9478444377449503
```

Figure 9: Training Output

6.2.3 Exporting Model

The output the export module is shown in figure 9. It outputs a JSON file representing the Random Forest parameters.



```
{"n_features": 17, "n_classes": 2, "classes": ["-1", "1"], "n_outputs": 1, "n_estimators": 10, "estimators
[{"type": "split", "threshold": "7 <= 0.5", "left": {"type": "split", "threshold": "14 <= -0.5", "left":
{"type": "split", "threshold": "7 <= -0.5", "left": {"type": "split", "threshold": "13 <= -0.5", "left":
{"type": "split", "threshold": "15 <= 0.0", "left": {"type": "split", "threshold": "12 <= 0.5", "left": {"type":
 "leaf", "value": [[93.0, 0.0]]}, "right": {"type": "split", "threshold": "0 <= 0.0", "left": {"type": "leaf",
"value": [[6.0, 0.0]]}, "right": {"type": "split", "threshold": "6 <= -0.5", "left": {"type": "split",
"threshold": "1 <= 0.0", "left": {"type": "leaf", "value": [[4.0, 0.0]]}, "right": {"type": "leaf", "value":
[[0.0, 1.0]]}}, "right": {"type": "leaf", "value": [[1.0, 0.0]]}}}}}, "right": {"type": "split", "threshold": "2
<= 0.0", "left": {"type": "leaf", "value": [[34.0, 0.0]]}, "right": {"type": "split", "threshold": "0 <= 0.0",
"left": {"type": "split", "threshold": "3 <= 0.0", "left": {"type": "split", "threshold": "6 <= 0.5", "left":
{"type": "split", "threshold": "12 <= -0.5", "left": {"type": "leaf", "value": [[18.0, 0.0]]}, "right": {"type":
 "split", "threshold": "11 <= 0.0", "left": {"type": "split", "threshold": "12 <= 0.5", "left": {"type":
"split", "threshold": "6 <= -0.5", "left": {"type": "leaf", "value": [[0.0, 2.0]]}, "right": {"type": "leaf",
"value": [[4.0, 3.0]]}}, "right": {"type": "leaf", "value": [[3.0, 0.0]]}}, "right": {"type": "leaf", "value":
[[2.0, 0.0]]}}}, "right": {"type": "split", "threshold": "8 <= 0.0", "left": {"type": "leaf", "value": [[1.0,
0.0]]}, "right": {"type": "split", "threshold": "12 <= -0.5", "left": {"type": "leaf", "value": [[3.0, 0.0]]},
"right": {"type": "split", "threshold": "12 <= 0.5", "left": {"type": "leaf", "value": [[1.0, 2.0]]}, "right":
{"type": "leaf", "value": [[1.0, 0.0]]}}}}}, "right": {"type": "split", "threshold": "12 <= -0.5", "left":
```

Figure 10: Model JSON

6.2.4 Plugin Feature Extraction

The 17 features extracted for the web page at the techcache.science are logged in to the console which is shown in figure 11. The features are stored as key value pairs and the values are encoded from -1 to 1 as discussed above.

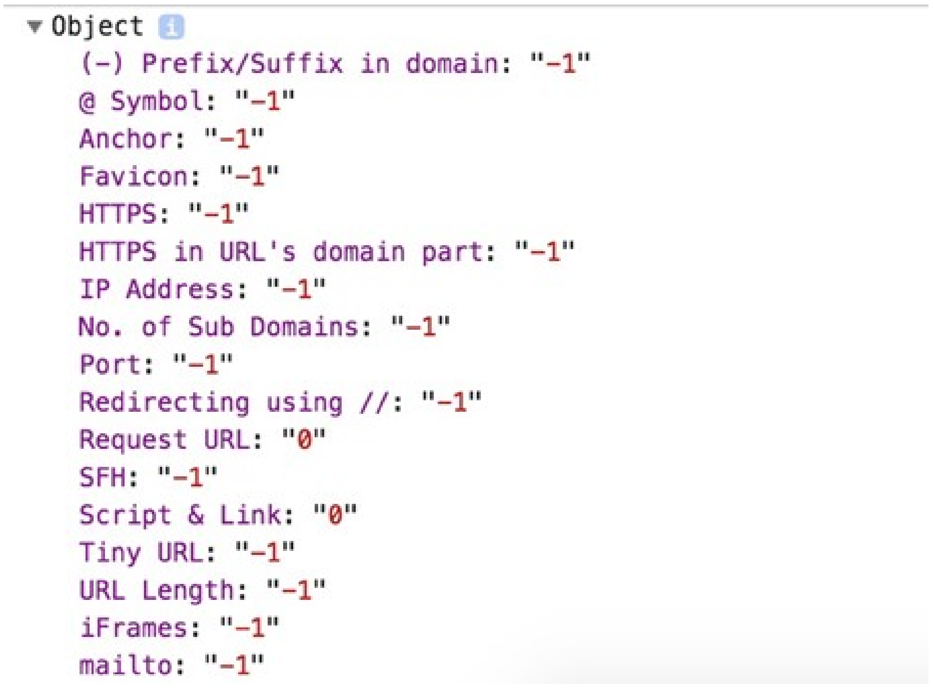

```
▼ Object
    (-) Prefix/Suffix in domain: "-1"
    @ Symbol: "-1"
    Anchor: "-1"
    Favicon: "-1"
    HTTPS: "-1"
    HTTPS in URL's domain part: "-1"
    IP Address: "-1"
    No. of Sub Domains: "-1"
    Port: "-1"
    Redirecting using //: "-1"
    Request URL: "0"
    SFH: "-1"
    Script & Link: "0"
    Tiny URL: "-1"
    URL Length: "-1"
    iFrames: "-1"
    mailto: "-1"
```

Figure 11: Web page Features

6.2.5 Classification

The output of the classification is shown right in the Plugin UI. Green circle indicates legitimate site and Light red indicates phishing.

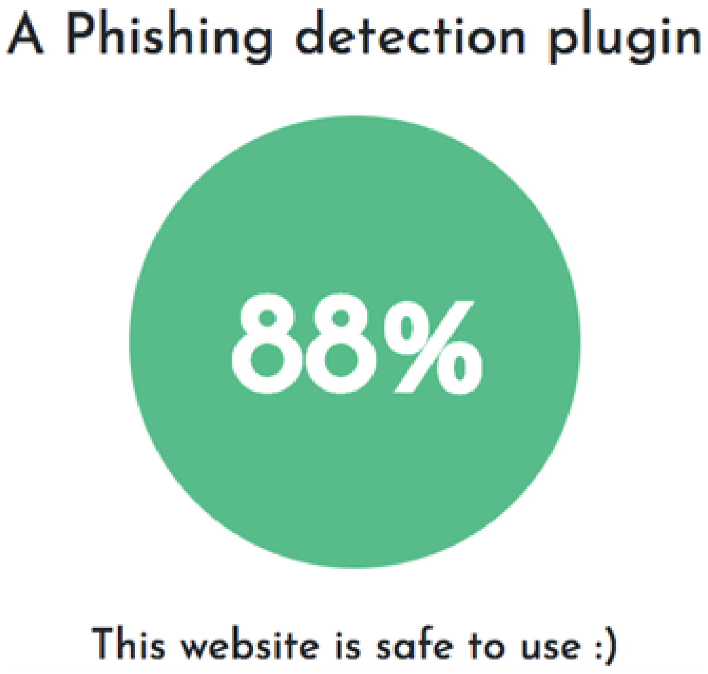


Figure 12: Classification Output

6.3 Performance Evaluation

The performance of the entire system is evaluated using the standard parameters described below.

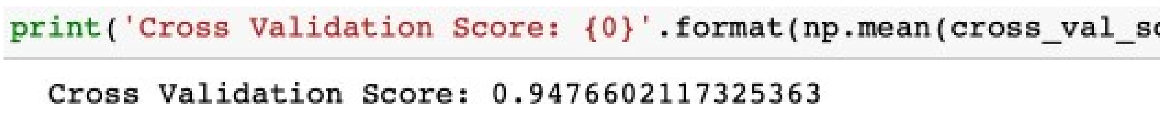
6.3.1 Cross Validation Score

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just re- peat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting.

To solve this problem, yet another part of the dataset can be held out as a so-called "validation set": training proceeds on the training set,after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k “folds”: A model is trained using k of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The score on 10 Fold cross validation is as below



```
print('Cross Validation Score: {0}'.format(np.mean(cross_val_sc
```

```
Cross Validation Score: 0.9476602117325363
```

Figure 13: Cross Validation Output

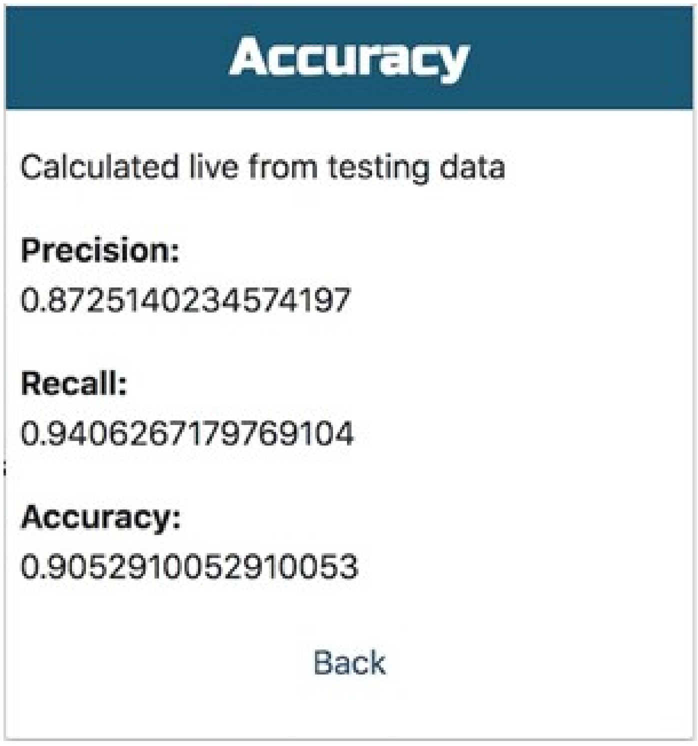
6.3.2 F1 Score

F1 score is a measure of a test’s accuracy. It considers both the precision and the recall of the test to compute the score: precision is the number of correct positive results divided by the number of all positive results returned by the classifier, and recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

The precision, recall and F1 score of the phishing classifier is calculated manually using JavaScript on the test data set. The results are shown in the figure 14.



Accuracy

Calculated live from testing data

Precision:
0.8725140234574197

Recall:
0.9406267179769104

Accuracy:
0.9052910052910053

Figure 14: Performance Measure

CHAPTER 7

Conclusion

7.1 Summary

This is a phishing website detection system that focuses on client side implementation with rapid detection so that the users will be warned before getting phished. The main implementation is porting of Random Forest classifier to JavaScript. Similar works often use web page features that are not feasible to extract on the client side and this results in the detection being dependent on the network. On the other side, this system uses only features that are possible to extract on the client side and thus it is able to provide rapid detection and better privacy. Although using lesser features results in mild drop in accuracy, it increases the usability of the system. This work has identified a subset of web page feature that can be implemented on the client side without much effect in accuracy. The port from python to JavaScript and own implementation of Random Forest in JavaScript further helped in rapid detection as the JSON representation of the model and the classification script is designed with time complexity in mind. The plugin is able to detect the phishing even before the page loads completely. The F1 score calculated on the test set on the client side is 0.886.

7.2 Future Scope

The classifier is currently trained on 17 features which can be in- creased provided that, they don't make the detection slower or result in loss of privacy. The extension can made to cache results of frequently visited sites and hence reducing computation. But this may result in pharming attack being undetected. A solution needs to be devised for caching of results without losing the ability to detect pharming. The classification in JavaScript can be done using Worker Threads which may result in better classification time. Thus a lot of improvements and en- enhancements are possible this system offers a more usable solution in the field of phishing detection.

Bibliography

[1] A. Subasi, E. Molah, F. Almkallawi, and T. J. Chaudhery, **“Intelligent phishing website detection using random forest classifier,”** 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Nov. 2017.

[2] **“UCI Machine Learning Repository: Phishing Websites Data Set,”** [Online]. Available: https://archive.ics.uci.edu/ml/datasets/ phishing websites.

[3] J.-H. Li and S.-D. Wang, **“PhishBox: An Approach for Phishing Validation and Detection,”** 2017 IEEE 15th Intl Conf on Depend- able, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), 2017.

[4] A. A. Ahmed and N. A. Abdullah, **“Real time detection of phishing websites,”** 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016.

[5] R. Aravindhan, R. Shanmugalakshmi, K. Ramya, and S. C., **“Cer- tain investigation on web application security: Phishing detection and phishing target discovery,”** 2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS), 2016.