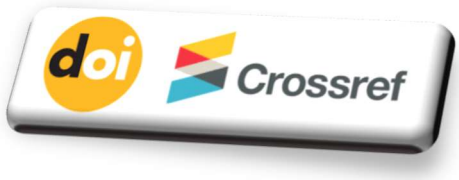
Journal, Advances in Mathematical & Computational Sciences
Vol. 9 No. 2, 2021
www.mathematics-computationaljournal.info



Journal of Advances in Mathematical & Computational Sciences
An International Pan-African Multidisciplinary Journal of the SMART Research Group
International Centre for IT & Development (ICITD) USA

Available online at https://www.isteams.net/ mathematics-computationaljournal.info
DOI: dx.doi.org/10.22624/AIMS/MATHS/V9N2P1
CrossREF Member Listing - https://www.crossref.org/06members/50go-live.html

A Random Forest Classifier-Based Low latent In-Browser Based Phishing Detection Plug-in Using Pattern Matching

[1]Longe, O.B, [2]Okunoye, A.A., [3]Okorejior, M. & [4]Asani, E.O.
Faculty of Computational Sciences & Informatics, Academic city University College, Accra, Ghana
[2]Williams College of Business, Xavier University, Cincinnati, Ohio, USA.
[3]School of IT & Computing, American University of Nigeria, Yola, Nigeria
Department of Computer Science, Landmark University, Omu-Aran, Nigeria
E-mails: olumide.longe@acity.edu.gh; okunoye@xavier.edu, mohammed.okorejior@aun.edu.ng; asani.emmanuel@lmu.edu.ng

ABSTRACT

We present a novel client-side In-browser phishing detection plugin that can automatically detect and warn users about phishing websites in real-time, using pattern matching based on the random forest classifier. Random Forest classifier is established in literature to perform considerably well in detecting phishing websites. The advantages of the proposed method include improved privacy of users' browsing data and performance in spite of low network latency. The system was implemented mainly, using JavaScript, while the Random Forest classifier was trained on the phishing websites dataset using python scikit-learn. The learned model parameters were exported in a portable format for the plug-in to be lightweight; this was in consideration of the expected low-latent processing power of the client machines. To the best of our knowledge, this is the first implementation of phishing website detection In-browser plugin without the use of external web services; the plugin with a one-time download of the learned model will be able to classify websites in real-time.

Keywords— In-browser, Phishing, Random Forest Classifier, Low Latency, Server-side, Client-side, Real-time

Longe, O.B, Okunoye, A.A., Okorejior, M. & Asani, E.O. (2021): A Random Forest Classifier-Based Low latent In-Browser Based Phishing Detection Plug-in Using Pattern Matching. Journal of Advances in Mathematical & Computational Science. Vol. 9, No. 2. Pp 15-32
DOI: dx.doi.org/10.22624/AIMS/MATHS/V9N2P2. Available online at www.isteams.net/mathematics-computationaljournal.

1. INTRODUCTION

Owing perhaps to its ubiquity and seemingly boundless potentials for communications, interconnectivity, transactions and so on, the internet is attractive to divergent tendencies, engendering both positive and negative activities. While internet technologies are at the cutting edge of great innovations and revolutionary findings, they unfortunately provide a veritable platform for criminals, to easily deploy and perpetrate criminal agenda, with global reach.

This has resulted in the alarming proliferation of cybercrimes, with consequences for both individuals and corporate organizations, estimated at hundreds of billions of dollars annually [1][2]. Phishing is arguably, one of the most prominent cybercrimes. Phishing is the attempt by criminals to surreptitiously obtain personal, sensitive credentials from unsuspecting users, more often than not, through the combination of technology and social engineering for malicious purposes [3][4]. Phishers, masquerading as genuine entities carry out their acts often through email spoofing or instant messaging to lure users to divulge vital, personal, often, financial information via a fake website, which nevertheless is identical in look and feel to its legitimate version. The phishing cycle begins with an email from an attacker mimicking the identity of a trusted entity often with a dubious revalidation exercise backed with a subtle threat or offer of reward aimed at compelling the user to click an embedded link that redirects to a fake website. The user is then required to fill supposed routine information which inadvertently stores users' information which may be used by the criminals to perpetrate illegal transactions later. Figure 1 and Figure 2 show samples of phishing email and 'fake versus real website' respectively.

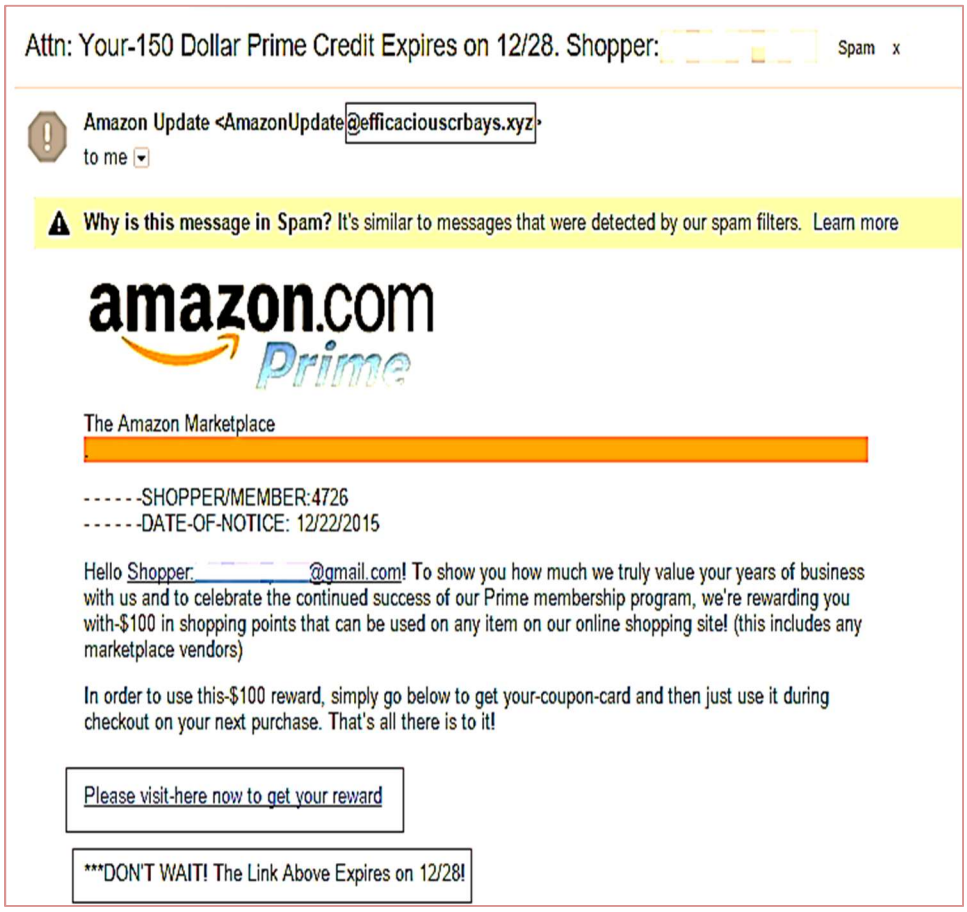


Fig. 1. Phishing Email Making A Bogus Claim Of Reward To An Unsuspecting Recipient (Source: Dascalescu, 2018)

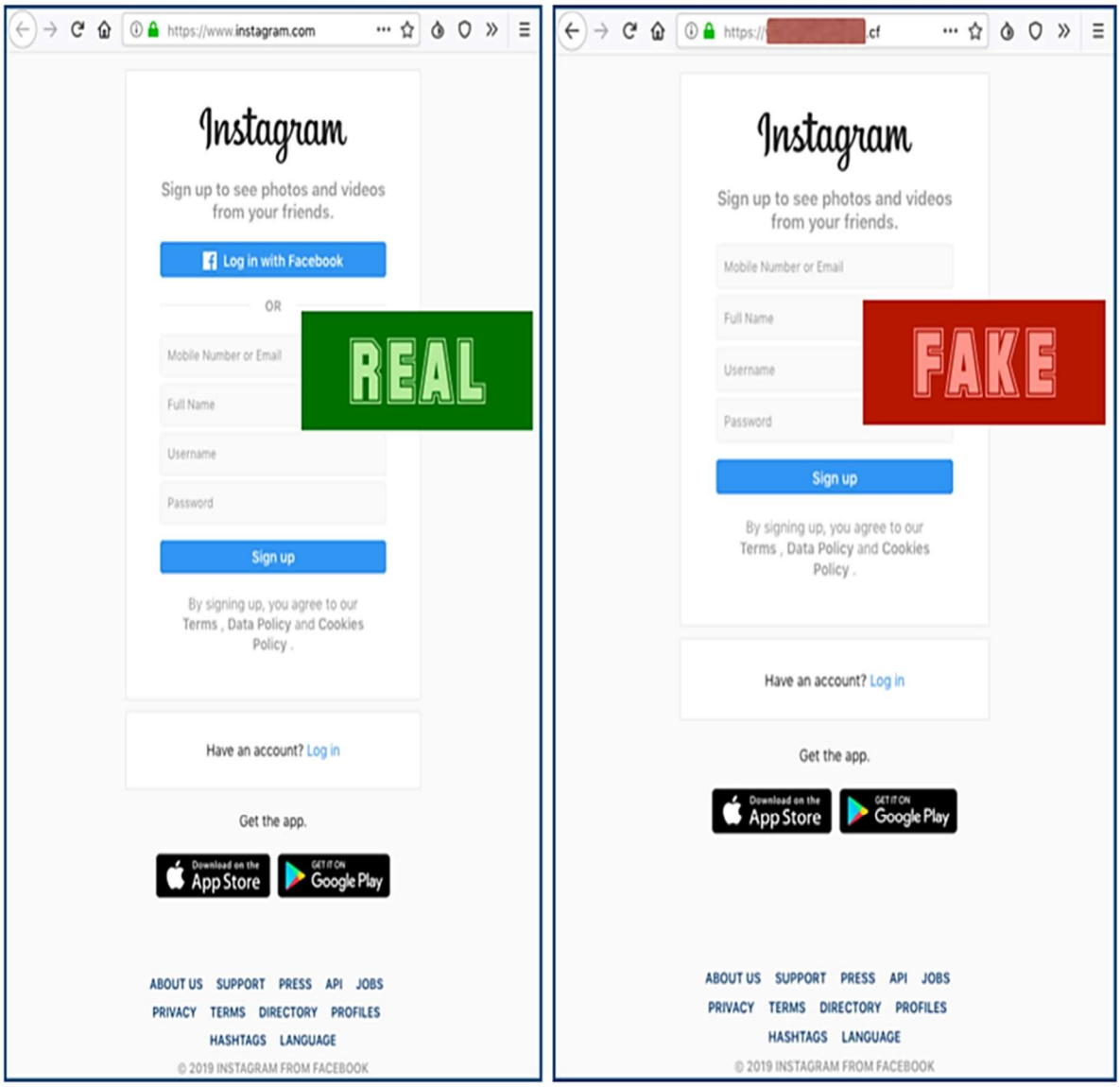


Fig. 2. Fake versus real website (Sophos https://nakedsecurity.sophos.com)

Phishing attacks pose serious risks to both individuals and corporate entities and have dire consequences on global security and the Economy [5]. Phishers continue to perfect means to outmaneuver even the knowledgeable and security conscious [6]; technology giants such as Google and Facebook lost about $100 million to phishing emails from hackers who impersonated as hardware vendor in 2017. The economic effect of phishing attack is enormous; report gathered over a period of five years by the FBI internet crime complaint center shows that financial loss occasioned by phishing attacks exceeds $12 billion globally [7].

In light of the rate of phishing crimes and the persistent efforts of phishers and considering the immense potential loss, the need to device improved, flexible means of safeguarding users' information has become pertinent. Thus, in this study, a novel client-side In-browser phishing detection plugin was developed. The developed system implements pattern matching based on the random forest classifier. The advantages of the proposed method include improved privacy of users' browsing data and performance in spite of low network latency. To the best of our knowledge, this is the first implementation of phishing website detection In-browser plugin without the use of external web services; the plugin with a one-time download of the learned model will be able to classify websites in real-time. In the remaining part of this article, a review of some selected literature, the research methodology, and design are presented. Results and system evaluation are also presented. The work closes with conclusions and recommendation for further work.

2. RELATED LITERATURE

Authors [8] highlighted some antiphishing techniques that have been deployed in research. They include both 'traditional' and 'computerized' means. The traditional methods identified law enforcement, education of users and other stakeholders. The computerized techniques include blacklists, filtering, Associative Classification, and rule induction as well as the use of machine learning approaches via different classification and model-based techniques. A variety of surveys and reviews of anti-phishing techniques have also been documented in the literature, all with the aim to provide better understanding and to enhance the development of better anti-phishing systems. Generally, anti-phishing techniques implementations are broadly categorized either as client-side implementations or server-side implementations as shown in figure 3.

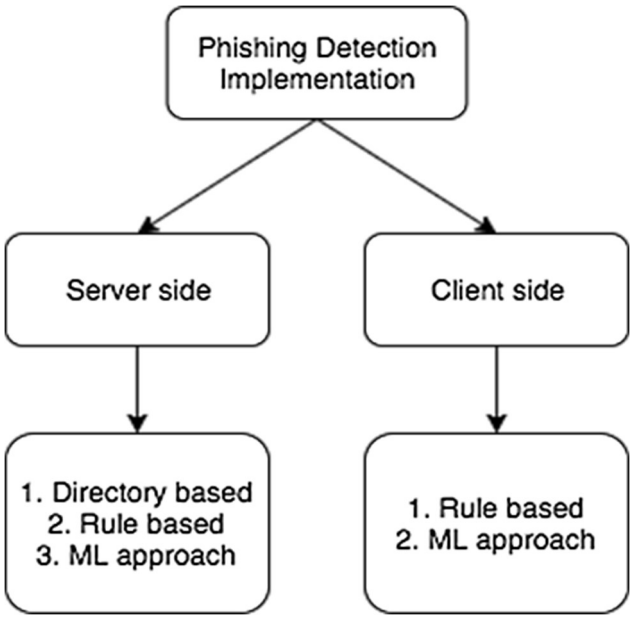


Fig. 3. Approaches to Phishing Detection

The directory-based approaches are deployed by using blacklists and whitelists. The principle of this technique is simple; reference blacklists containing URLs of phishing sites and/or whitelists which are directories of legitimate websites. These are then compared to the URL to be visited either at the client-side or from a remote server [9]. A survey by authors [10] examined blacklist and whitelist-based anti-phishing browser plug-ins. A directory based anti-phishing developed by [11] accepts URLs entered by users as input and checks it against the Google's blacklist of phishing websites using Google Safe Browsing API. It sends an HTTP API pull request and determines whether or not the URL is a phishing URL based on the response. A similar system by [12], equally deploys the Google Safe Browsing API for phishing URL apprehension. A popup notifies the user if the URL is a phishing URL. The system developed by [13] deployed both the whitelist and blacklist of URLs. The system compares the input URL first with the whitelist and checks the blacklist if no match was found. The directory-based techniques' major drawback is its inability to detect zero-day attacks because it has to rely on delayed updates.

The rule-based approach relies on a knowledge base to make inferences on whether or not a URL is a phishing URL or not. Authors [14] developed a rule-based data mining system that makes inference based on set rules and features using the C4.5 classification algorithms. Their work outperformed some existing systems such as RIPPER, PRISM, and CBA algorithms in terms of accuracy. A rule-based system developed by [15] made use of '*Levenshtein Distance'* for string matching to find the relationship between the content and the URL of a webpage and used SVM for classification. Work by [16] on the detection of phishing attacks using genetic algorithm used a rule that is generated by a genetic algorithm for phishing detection. An evaluation of their system showed promising results. Authors [17], in their bid to identify the most suitable features, used five algorithms, namely, C4.5, Ripper, Part, Prism, and CBA. The features were categorized into six classes namely "*URL and domain identity*", "*security and encryption*", "*source code and Javascript*", "*page style and contents*", "*web address bar and social human factor"*. They assigned weights to each of these feature sets, and fuzzy rules were created for phishing website detection. Although rule-based approaches support easier implementation on client-side, they have been known to fall short in comparison with Machine Learning based approaches.

Authors [18] developed an intelligent phishing website detection system using the Random Forest classifier. Multiple data mining methods were implemented in order to arrive at the correct classification. The systems were evaluated based on accuracy, (ROC) curves (AUC) and F-measure. Results showed that Random Forest outperformed other methods by achieving an accuracy of 97.36%. They also observed that the computing time of the Random Forest algorithm was considerably fast.

In a study reported by [19], 'PhishBox' was developed to correctly detect phishing websites. The approach uses a 'two-step' detection model for improved performance. The first was the development of an ensemble model to validate the phishing corpus and apply active learning. The ensemble's performance was very satisfactory with an accuracy of 95%. The second stage accepts the validated data as input for training. The system was able to reduce false positive by a massive average of 43.7%. Authors [20] developed a Real time phishing detection system. They were able to come up with a detection mechanism that could detect various types of phishing attacks and maintained a low rate of false alarms. Authors [21] developed an antiphishing system based on concept features. An Additional domain top-page similarity feature was applied to the system. The evaluation result in terms of F-measure was up to 0.9250, with 7.50% of error 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.

Since most of the phishing websites are short-lived, the directory approach cannot always keep track of all, including new phishing web- sites. So, the problem of detecting phishing websites can be solved in a better way by machine learning techniques.

3. DESIGN

Use Case

The use case diagram of the entire system is shown in figure 4. The user installs the plugin and then continue his/her normal browsing behavior. The plugin automatically checks the browsing pages for phish and warns the user of the same. Pre-condition: The user visits a website and have plugin installed

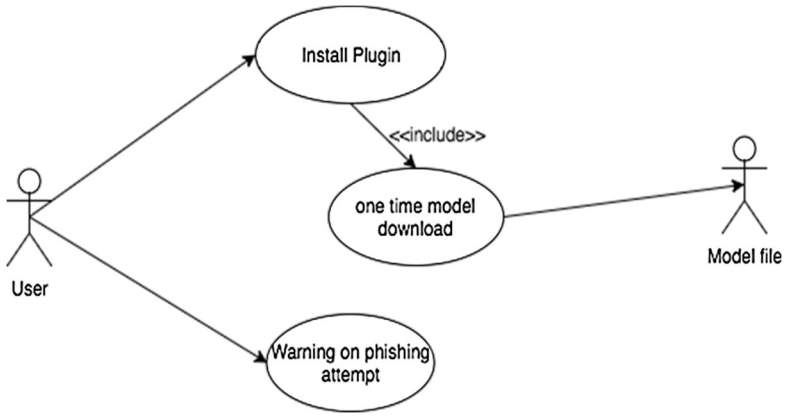


Fig. 4. Use case diagram of the system

The sequence of interactions between the user and the plugin are shown in the figure 5.

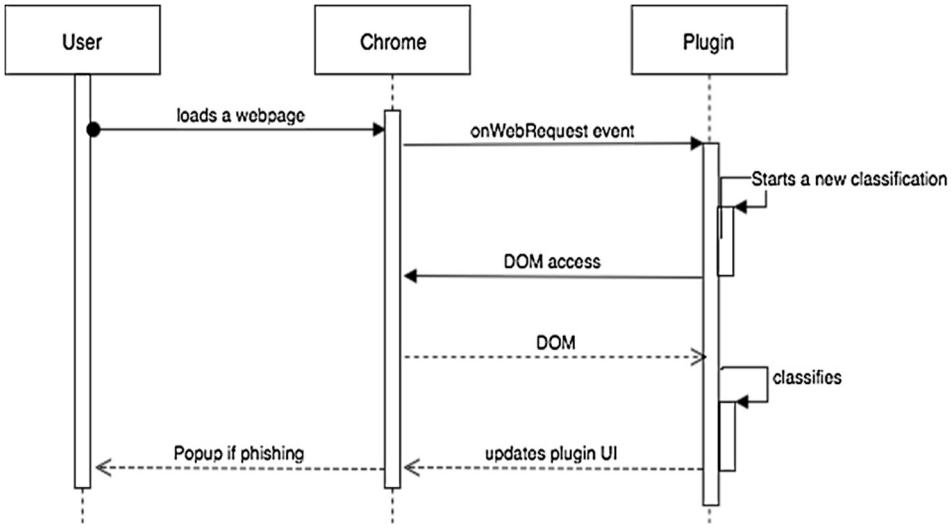


Fig. 5. System Sequence diagram

System Architecture

The architecture diagram of the entire system is shown in the figure 6. The Random Forest classifier was trained on a phishing sites dataset using python scikit-learn. Random Forest algorithm is an ensemble learning technique and thus an ensemble of ten decision tree estimators was used. Each decision tree follows CART algorithm and tries to reduce the Gini impurity:

$$Gini(E) = 1 - \sum_{j=1}^{c} \quad p_j^2 \qquad \text{.....(1)}$$

The cross-validation score is also calculated on the training data. The F1 score is calculated on the testing data. Then the trained model is exported to JSON using the next module. The Random Forest classified was formatted as a JavaScript Object Notation (JSON) and exported. A browser script which uses the exported model JSON to classify the website being loaded in the active browser tab was then implemented. The classifier identifies whether the site is phishing or legitimate. The system sends a warning prompt to the user in the event of phishing.

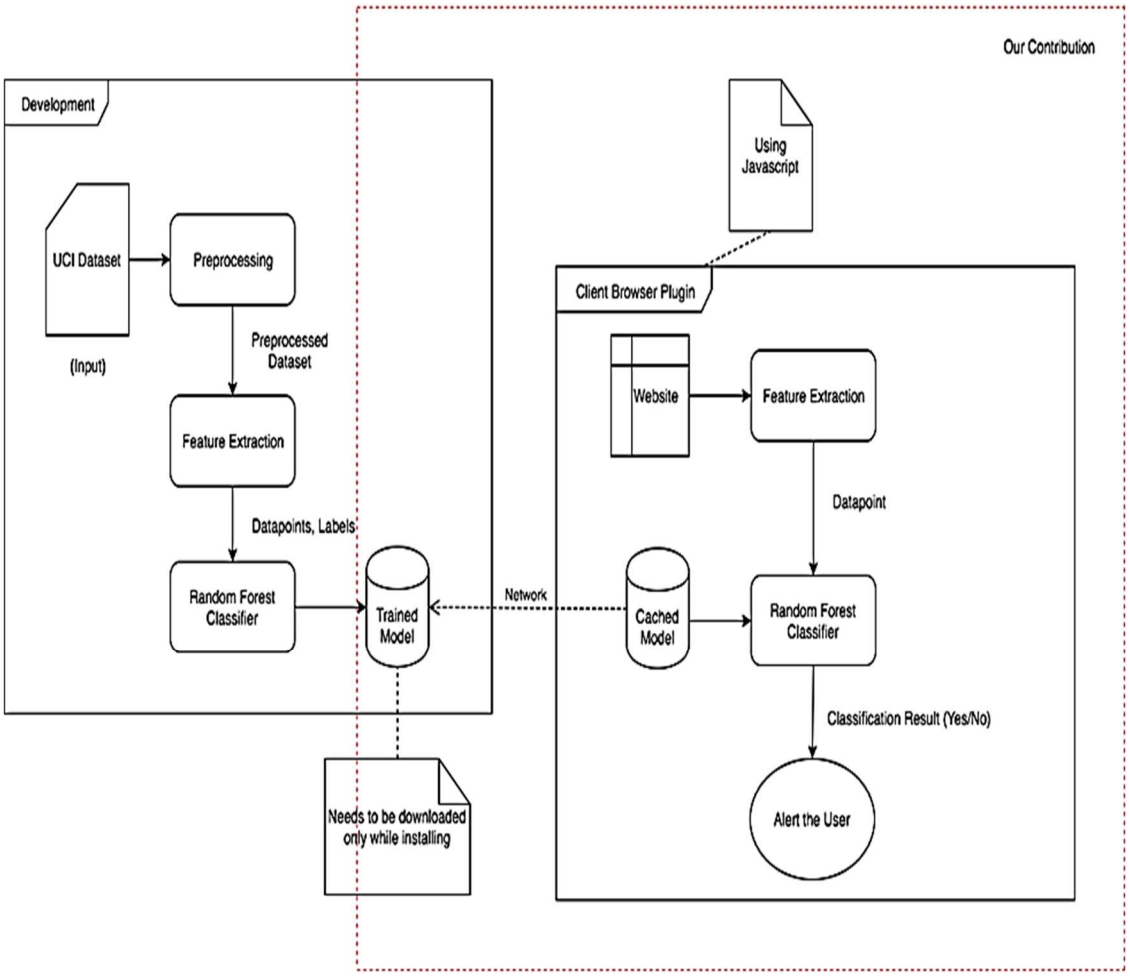


Fig. 6. System Architecture

The dataset raff file was loaded using python raff library and 17 features were chosen from the existing 30 features. Features were selected on basis that they can be extracted completely offline on the client side without being dependent on a web service or third party. The dataset with chosen features was then separated for training and testing. Then the Random Forest was trained on the training data and exported to the above mentioned JSON format. The JSON file is hosted on a URL. The client-side chrome plugin was made to execute a script on each page load, and it starts to extract and encode the above selected features. Once the features are encoded, the plugin then checks for the exported model JSON in cache and downloads it again in case it is not in the cache. With the encoded feature vector and model JSON, the script can run the classification. Then a warning is displayed to the user, in the case that the website is classified as phishing. The entire system was designed as low latent so that the detection will be rapid.

UI Design

A simple and easy to use User Interface was designed for the plugin using HTML and CSS. The UI contains a large circle indicating the percentage of the legitimacy of the website in active tab. The circle also changes its color with respect to the classification output. Below the circle, the analysis results containing the extracted features are displayed in the following color code.

a) Green - Legitimate
b) Yellow - Suspicious
c) Red - Phishing

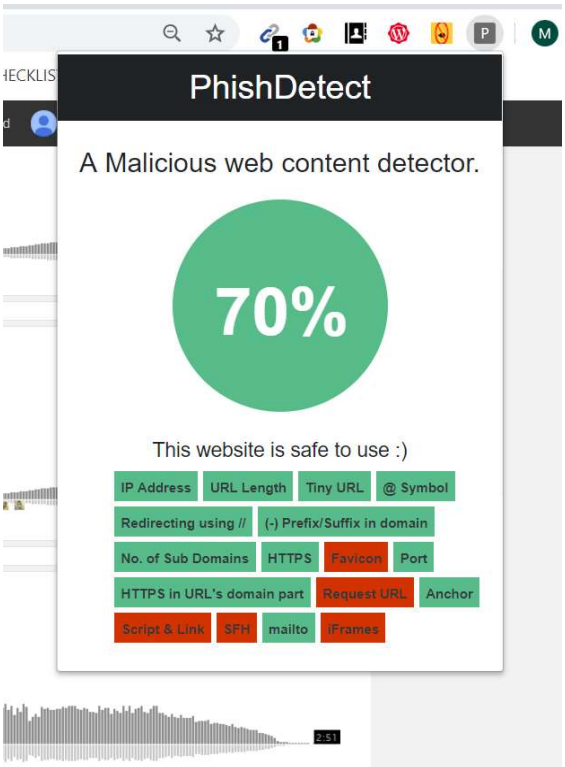


Fig. 7. System UI

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 in a separate screen. The UI is shown in figure 7.

Class Diagram

This class diagram depicts the functions of various modules in the system. It also shows the interaction between the modules of the system thereby providing a clear idea for implementation.

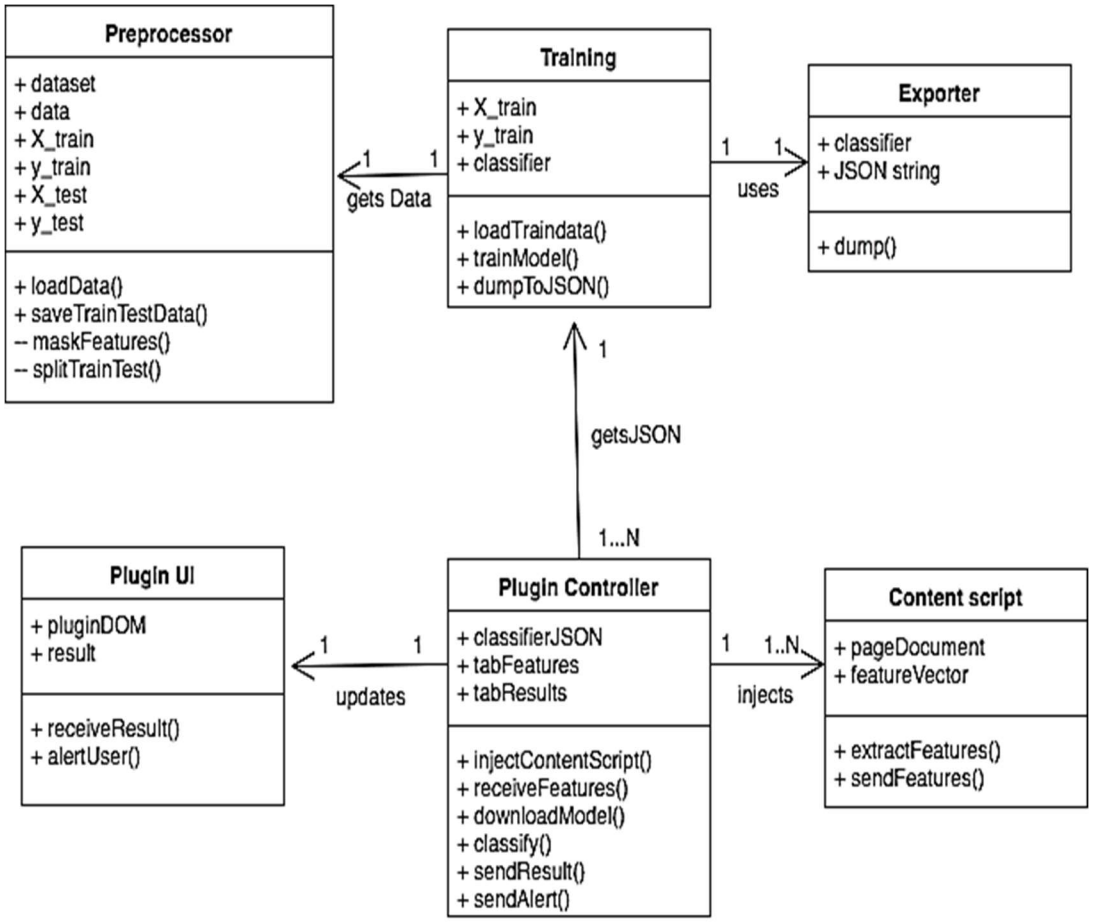


Fig. 8. The System's Class Diagram

Data Preprocessing

The dataset was downloaded from UCI repository and loaded into a numpy array. The dataset consists of 30 features, which needs to be reduced so that they can be extracted on the browser. Each feature was experimented on the browser so that it would be feasible to extract without using any external web service or third party. Based on the experiments, 17 features were chosen out of 30 without much loss in the accuracy on the test data. More number of features increases the accuracy and on the other hand, reduces the ability to detect rapidly considering the feature extraction time. Thus, a subset of features is chosen in a way that the tradeoff is balanced.

The feature include IP address, Degree of subdomain, Anchor tag ***href*** of domains, URL length, HTTPS or not, Script & link tag domains, URL shortener, Favicon domain, Empty server form handler, '@' in URL, TCP Port, Use of mailto, Redirection with '//', HTTPS in domain name, Use of iFrame, "-" in domain, Cross domain requests. The dataset was split into training and testing set at ratio 7:3 respectively. Both the training and testing data were saved to disk.

Classification

Every machine learning algorithm learns its parameter values during the training phase. In Random Forest, each decision tree is an independent learner and each decision tree learns node threshold values and the leaf nodes learn class probabilities. Thus, 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 and so on. Furthermore, it contains an array Random Forest JSON structure in which each value is an estimator represented in JSON. Each decision tree is encoded as a JSON tree with nested objects containing threshold for that node and left and right node objects recursively.

The features need to be extracted and encoded for each webpage in real-time while the page is being loaded. A content script was used so that it could access the DOM of the webpage. The content script is automatically injected into each page while it loads. The content script is responsible for collecting the features and then send them to the plugin. The main objective of this work is to eradicate the need for any external web service and have features independent of network latency, thereby achieving rapid extraction. All these were made sure of, while developing techniques for the extraction of features. Once a feature is extracted it is encoded as values {-1, 0, 1} based on the following notation.

-1 - Legitimate
0 - Suspicious
1 - Phishing

The feature vector containing 17 encoded values was passed on to the plugin from the content script. The feature vector obtained from the content script is ran through the Random Forest algorithm for classification. The Random Forest parameter JSON is downloaded and cached in disk. The script tries to load the JSON from disk and incase of cache miss, the JSON is downloaded again. A JavaScript library was developed to mimic the Random Forest behavior using the JSON by comparing feature vector against the threshold of the nodes. The output of the binary classification is based on the leaf node values and the user is warned if the webpage is classified as phishing.

Trade-off

One of the uniqueness of this study lies in balancing the tradeoff between accuracy and rapid detection. Choosing a subset of features that will make the detection fast and at the same time without much drop-in accuracy. This process includes:

- Porting of scikit-learn python object to JavaScript compatible format.
- Reproducing the Random Forest behavior in JavaScript reduced the accuracy by a small margin.
- Identifying and removing features that are not feasible to extract without using an external web service.
- Maintaining rapid detection before the user submit any sensitive information.

Development

The system is 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 input and output to each module of the system is described as follows:

- **Preprocessing**: This module takes the downloaded dataset in arff format and 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 the 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 recursively generates its JSON representation which is written to file in disk.
- **Plugin Feature Extraction:** This module takes a webpage as input and generates 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 model module and then gives a boolean output which denotes whether the webpage is legitimate or phishing.

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

Algorithm 1: 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
```

Algorithm 2: 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
```

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

4. RESULTS AND DISCUSSION

The test set consists of data points separated from the dataset by ratio 70:30. Also the plugin was tested with websites that are listed in phishTank. New phishing sites were also added to PhishTank as soon as they were found. It should be noted that the plugin was able to detect zero-day phishing sites. The results of this module testing as well as the testing of the entire system are summarized as follows:
The output of the preprocessing module is shown in figure 9.

```
~/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
```

Fig. 9. Preprocessing Output

The output the training module is shown in Figure 10

```
~/D/m/phishing_detector  *+  backend/classifier  python3 training.py              Fri Oct 26 13:03:31 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: 0.9455923597113163
Accuracy: 0.946940060295448
```

Fig. 10. Training Output

The output the export module is shown in Figure 11. It outputs a JSON file representing the Random Forest parameters

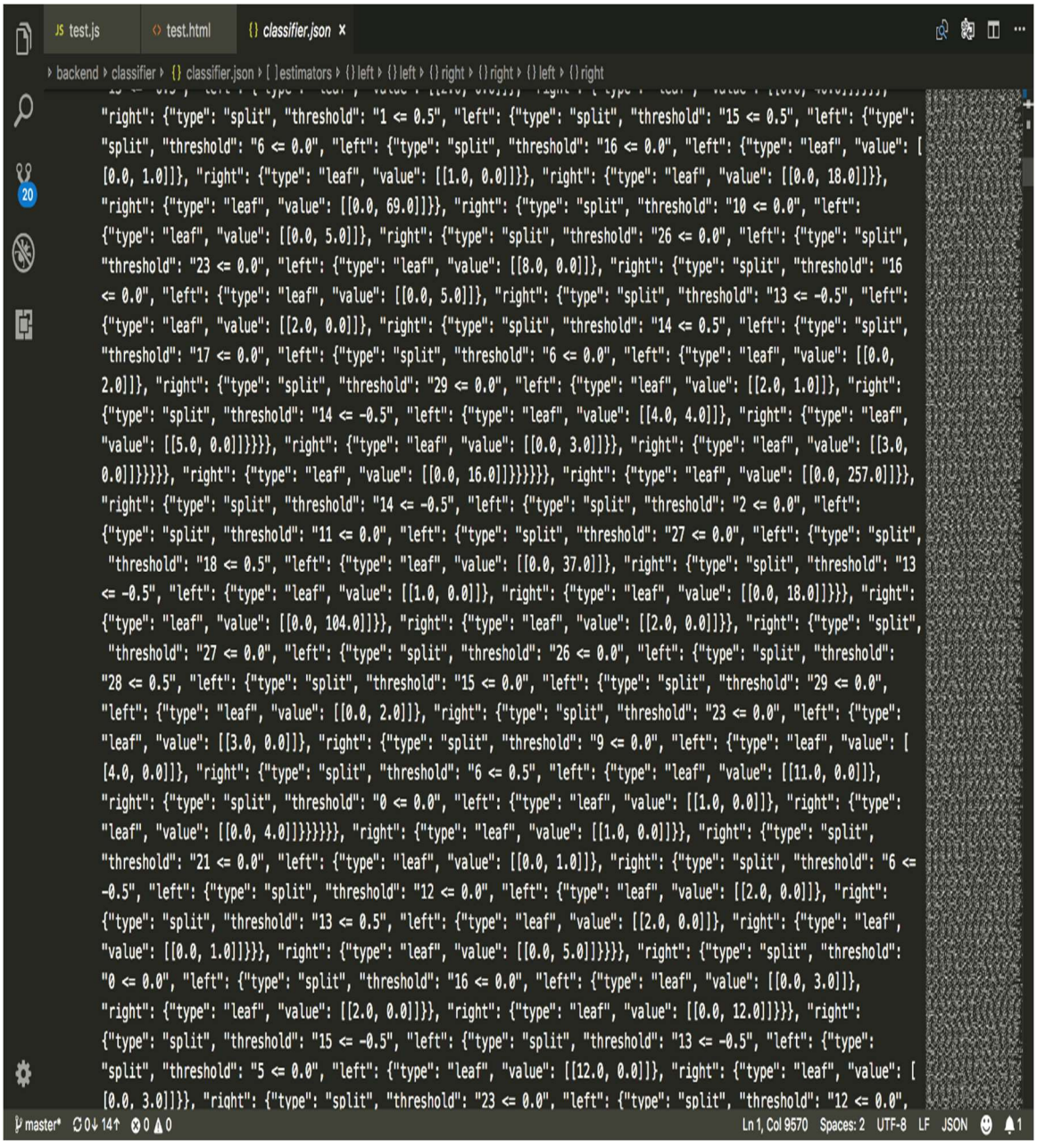


Fig. 11. Export Module JSON Output

The 17 features extracted for the webpage at the specific URL were logged in to the console which is shown in Figure 12. The features were stored as key value pairs and the values were encoded from -1 to 1 as discussed in section 5.

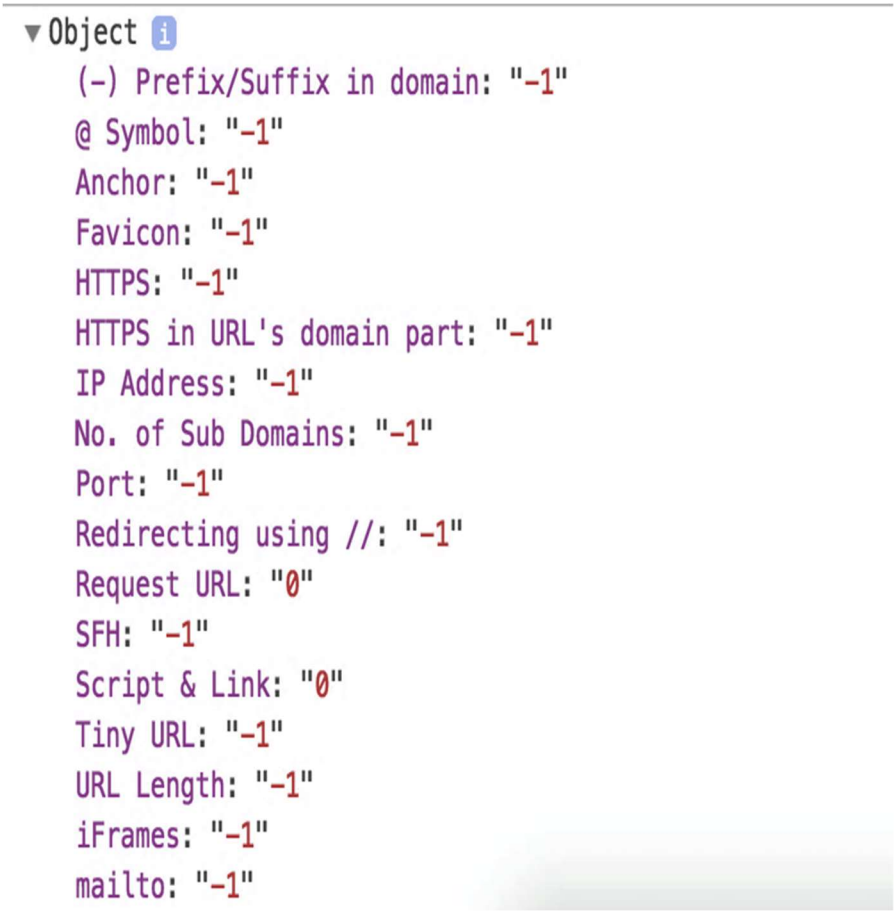


Fig. 12. Webpage Features

The output of the classification is shown in Fig 13. Green circle indicates legitimate site and Light red indicates phishing. The pictured site has a low trust value and the light red circle indicates phishing.

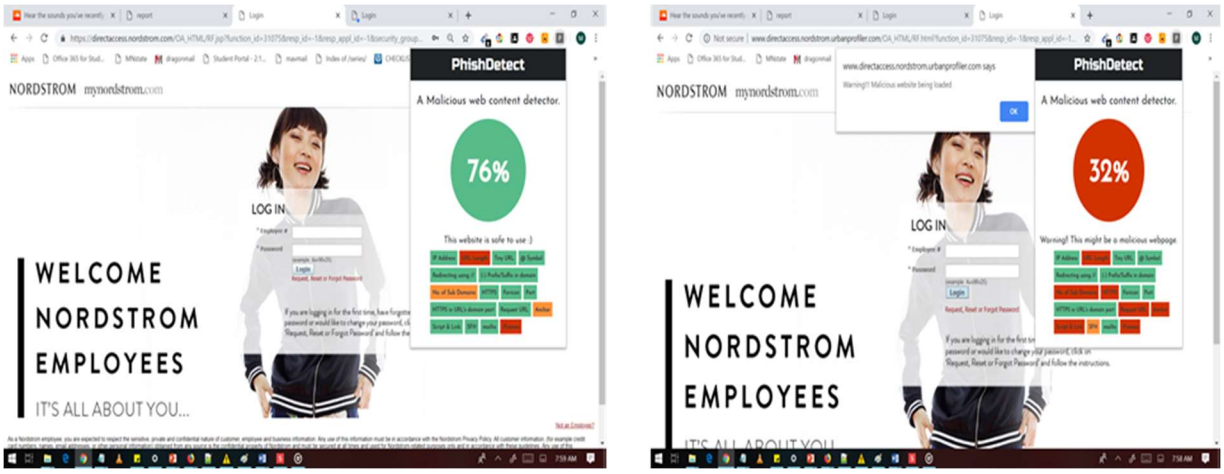


Fig. 13. Test Output

5. CONCLUSION

Summary

In this study, we presented the design and development of 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 the porting of the Random Forest classifier to JavaScript. Whereas, similar works often use webpage features that are not feasible to extract on the client-side and thus making them dependent on the network. This system uses only features that are possible to extract on the client-side and thus it can provide rapid detection and better privacy. Although using lesser features results in a mild drop in its accuracy, it increases the usability of the system. This work has identified a subset of webpage features that can be implemented on the client-side without much effect on its accuracy. The port from python to JavaScript and the implementation of the Random Forest algorithm in JavaScript further helped in the rapid detection as the JSON representation of the model and the classification script was designed with time complexity in mind. The plugin was able to detect the phishing webpage even before the page loads completely.

Drawbacks of the System

The system has a lower accuracy than the state-of-the-art, but it is more usable and the trade-off between accuracy and rapid detection was handled well enough. The chrome extension API restrictions have a small effect on the plugin. Since the features are extracted in the content script, which is injected on page load, this plugin cannot prevent a malicious JavaScript code from executing. Further, the accuracy reduces while porting from python to JavaScript and this needs to be investigated. JavaScript does not support multithreading and browser executes only JavaScript. Thus, the classification can't be made faster by using parallel threads. Currently, the results are not cached on the plugin and it's computed repeatedly even for frequently visited sites.

6. FUTURE RESEARCH DIRECTIONS

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

Acknowledgment
We gratefully acknowledge the support of the Society for Multidisciplinary & Advanced Research Techniques (SMART) and Landmark University, SDG 11 for access to research repositories, literary materials and useful insights from their partners.

REFERENCES

[1] Hong J. (2012). The state of phishing attacks. Communications of the ACM, 55 (1), 74-81. doi:10.1145/2063176.2063197
[2] Smith Z.M. & Lewis E.A., (2020). The Hidden Cost of Cybercrime. Retrieved June 3, 2021. url: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-hidden-costs-of-cybercrime.pdf.
[3] Asani & Adegun (2014). Maximum Phish Bait: Towards Feature Based Detection of Phishing using Maximum Entropy Classification Technique. In proceedings of International Conference on Science, Technology, Education, Arts, Management and Social Sciences iSTEAMS.
[4] Emmanuel Oluwatobi Asani, Adebayo Omotosho, Paul Ashante Danquah, Joyce Ayoola, Peace Ayegba & Olumide B. Longe (2021). A Maximum Entropy Classification Scheme for Phishing detection using Parsimonious Features. TELKOMNIKA Telecommunication, Computing, Electronics and Control 19(5). http://dx.doi.org/10.12928/telkomnika.v19i5.15981
[5] Islam R. & Abawajy J., (2013). A multi-tier phishing detection and filtering approach. Journal of Network and Computer Applications, 36(1), 324-335. doi>10.1016/j.jnca.2012.05.009
[6] Lucky, R. W. (2011). Clickphobia [Reflections]. IEEE Spectrum, 48(1), 25. doi:10.1109/MSPEC.2011.5676377
[7] Federal Bureau of Investigation (2018). Business E-mail Compromise The 12 Billion Dollar Scam. Retrieved 22 Feb., 2020. url: https://www.ic3.gov/media/2018/180712.aspx
[8] Qabajeh, I., Thabtah, F. & Chiclana, F., (2018). A recent review of conventional vs. automated cybersecurity anti-phishing techniques. Computer Science Review 29, 44-55. https://doi.org/10.1016/j.cosrev.2018.05.003
[9] Varshney G., Misra M. & Atrey P.K., (2016). A survey and classification of web phishing detectionschemes. Security Comm. Networks 9, 6266–6284. DOI: 10.1002/sec.1674
[10] Li L., Berki E., Helenius M. & Ovaska S., (2014). Towards a contingency approach with whitelist- and blacklist-based anti-phishing applications: what do usability tests indicate? Behaviour & Information Technology 33(11), 1136–1147.
[11] Google (2014). Safe browsing API-developer guide V3. Retrieved 22 Feb., 2020. url: https://developers.google.com/safe-browsing/developers_guide_v3
[12] Firefox (2014). How does built-in phishing and malware protection work?. Retrieved 10 Oct, 2019. Url: https://support.mozilla.org/en-US/kb/how-does-phishing-and-malware-protection-work
[13] Krishnamurthy B, Spatscheck O, Van Der Merwe J & Ramachandran A (2009). Method and apparatus for identifying phishing websites in network traffic using generated regular expressions. Retrieved 22 Jan, 2020. Url: https://patents.google.com/patent/US8307431
[14] Mohammad R.M., Thabtah F. & McCluskey L., (2014). Intelligent rule-based phishing websites classification. IET Information Security 8(3), 153–160.
[15] Moghimi M., & Varjani A.Y., (2016). New rule-based phishing detection method. Expert Systems with Applications 53, 231–242.
[16] Shreeram, V., Suban, M., Shanthi K.P. & Manjula K., (2010). Anti-phishing detection of phishing attacks using genetic algorithm. In proceeding of International Conference on Communication Control and Computing Technologies, 447-450. doi: 10.1109/iccct.2010.5670593.
[17] Aburrous M., Hossain M.A., Dahal K. & Thabtah F., (2010). Intelligent phishing detection system for e-banking using fuzzy data mining. Expert Systems with Applications. 37(12), 7913–7921.
[18] Subasi A.E., Molah F., Almkallawi & Chaudhery T.J., (2017). Intelligent phishing website detection using random forest classifier. In proceedings of International Conference on Electrical and Computing Technologies and Applications (ICECTA), 1-5. doi: 10.1109/ICECTA.2017.8252051

[19] Li J. & Wang S., (2017). PhishBox: An Approach for Phishing Validation and Detection. In proceeding of IEEE 15th Intl Conf on Dependable, 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), 557-564. doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.101

[20] Ahmed A.A. & Abdullah N.A., (2016). Real time detection of phishing websites. In proceedings of IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 1-6. doi: 10.1109/IEMCON.2016.7746247

[21] Sanglerdsinlapachai N. & Rungsawang A., (2010). Using Domain Top-page Similarity Feature in Machine Learning-Based Web Phishing Detection. In proceedings of Third International Conference on Knowledge Discovery and Data Mining, Phuket, 187-190. doi: 10.1109/WKDD.2010.108