Phishing Detection in Browsers using Machine Learning

Tanmay Naik
Khoury College of Computer Sciences
Northeastern University
Boston, US
naik.t@northeastern.edu

Nithin Gangadharan
Khoury College of Computer Sciences
Northeastern University
Boston, US
gangadharan.n@northeastern.edu

***Abstract*—Phishing is a cybercrime in which a target visits a website that is posing as a legitimate application, to lure individuals into providing sensitive data such as - banking and credit card details, and passwords. An unsuspecting user can click a link in an email or social media platform, and be led to a phishing website, which may lead to fraud and identity theft. Phishing is a widespread attack that still does not have a concrete solution.**

This report proposes a solution for the protection of end users through a browser extension while comparing various Machine Learning approaches to identify phishing websites. Rule-based approaches can be limiting since there are new phishing websites created everyday. Machine Learning based approaches aim to solve that, with the trade-off being that results will be probabilistic rather than absolute. Experiments described in this report use a combination of these methods, which results in a model that serves the purpose well and requires little overall maintenance in the long run.

I. Introduction

With the recent advancement in various cybersecurity technologies, the weakest link in cybersecurity happen to be the end users. Attackers utilize phishing which exploits naivety of users to trick them into handing out sensitive information. This poses a great risk not only to the users themselves but the organizations and institutions of which they are a part of. According to recent research from Proofpoint, 75% of organizations around the world experienced a phishing attack in 2020, and 74% of attacks targeting US businesses were successful.[1]

Most of the software interactions between users and organizations happen through websites on computers/laptops. Forcing users to install a custom application to be run is a rarity due to the convenience a browser provides. Web-based security for browsers are crucial in current day and Phishing is one of the crimes that still exists.

Apart from increasing security awareness among users, the tools which complement that awareness to help users make safe decisions must be developed. This report proposes and demonstrates a Chromium-based browser extension to help mitigate the risk of phishing while browsing the web. The central idea of the browser extension is to notify the user whenever they open any *potential* phishing website.

The solution also includes a Python web server which utilizes various Machine Learning classification techniques to determine the legitimacy of the webpage in question. The web server takes in a URL and returns a boolean value indicating if the given URL is part of a potential phishing attempt.

The browser extension monitors each URL that the user visits, and tries to determine if the URL is malicious with the help of the web server. The web server exposes a REST API which is consumed by the extension for communication. The same API can also be reused as-is to implement a similar phishing detection in a different context like a network-level application or in a mobile application like Android.

The server implements both a mixture of rule based approach and Machine Learning classification techniques. The rule based approach is useful to weed out obvious URLs and is rather an inexpensive operation. The Machine Learning classification techniques is more expensive to do, but helps predict whether a URL can be a phishing site.

II. Related Work

a) PhishDetector[2]: This is a Chromium extension which has set out to solve exactly the same problem as this project. Based to its description on the website and the analysis of its behavior, it can be concluded that this extension uses a rule-based system to determine if a webpage is a phishing attempt. It also seems to be particularly accurate when it comes to identifying illegitimate banking pages. Even though rule-based systems are great for detecting simple phishing attempts, they are not ideal for more sophisticated phishing attempts. Rule-based systems are also inherently complex to maintain - adding and modifying rules over time makes the system more complex and unmanageable with time. They also demand more human intervention to define the rules and for their maintenance. Using Machine Learning in lieu of rule-based system gets rid of many of these limitations. As Machine Learning is data centric, it doesn't require managing complex rules and makes it straightforward to tweak the algorithms.

b) Cloudphish[3]: Cloudphish is a phishing detector for web-based email software. It monitors all emails received by a user and checks each email for a phishing attack. Having a paid subscription model, it offers a *decent* service. But the major limitation it has is that it works only with the email inbox. Even though many of the phishing attacks are carried over through email, phishing is as prominent on social media

and messaging apps in this age. And that calls for a solution which monitors all the web activity to identify phishing attacks regardless of their method of delivery.

There are various other browser extensions which virtually have the same limitations as the aforementioned solutions[4][5][6] As their limitations are encompassed in the discussion of other solutions above, their detailed discussion has been omitted for brevity.

To summarize, there are various browser plugins consisting of rule-based systems, simple whitelists-blacklists and some even making use of Machine Learning and Artificial Intelligence. But there needs to be a solution which utilizes all available phishing detection methods to protect the average internet user from criminals.

III. Approach

The architecture of this project consists of two primary components: The browser extension and the web server.

a) Browser Extension: The extension is developed for Chromium-based browsers using JavaScript with HTML and CSS. Therefore it is compatible with any Chromium implementation including Google Chrome, Microsoft Edge, Brave, etc. The extension monitors each web page that is visited by the user and fetches the URL of that webpage. It then communicates with the web server which tells if the given URL is part of a phishing attempt or not. The user is notified with the results of the analysis based on the response from server. The extension will stay silent in the background while the user is visiting websites that it has deemed safe. It only *bothers* them when there is a potential of phishing on the site they are currently visiting help them make a safer decision.

b) Python Web Server: The web server has an REST endpoint which takes in a URL and uses Machine Learning techniques to classify it into categories. It extracts relevant "features" from the URL and feeds them to *Classification Models* to determine the legitimacy of the URL. This process is expanded upon further down in this section.

Classification is a process of categorizing a given set of data into classes. In this case, there would be two classification labels: "spam" and "not spam". And the input data would be values of various features of the URL which are deemed effective for high quality classification of any URL into one of the classes.

The process of creating a classification model consists of two primary stages. The first is the training stage where a classifier is fed a large amount of input with along with their respective class labels. That creates a classification model which is given a set of inputs without their respective labels to let it classify each input to a class. That constitutes the second stage where the correctness of the newly created classification model is compared against the actual labels. The input set given to the model for the second stage is often referred to as *testing dataset* and the data used for the first stage is called *training dataset*. As a common practice, the dataset on hand is split into training and testing datasets for creation and testing of the classification models, respectively.

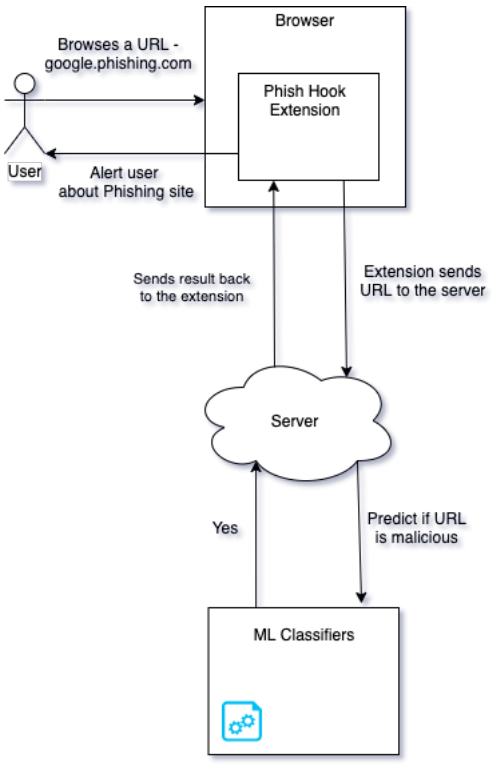

Fig. 1. Architecture overview of the application

The classifiers used in this project are described below:

1) Decision Trees: Decision Trees belong to the family of supervised machine learning algorithms. Decision trees classify the input by running them down the tree from the root node to some leaf node, whereas the leaf node provides the classification of the input.
2) K-Nearest Neighbors (KNN): The KNN algorithm assumes that similar things are *near* to each other. Based on this assumption, it classifies all nearby data points into one. Then, it classifies the given input by locating N-nearest neighbors and finding mode of their labels which is predicted to be the label of the input set.
3) Random Forests: Random Forests consist of a large number of Decision Trees that operate as an ensemble. Each tree in a forest classifies the given input set to a label and the label with the highest number of votes is considered as the prediction of the Random Forest.
   The individual trees in a random forest are relatively *uncorrelated*, and they perform better as an ensemble than they would on their own. To put it in layman's terms, the trees safeguard each other from their individual errors.

The following section describes the datasets used for training and testing the models with the aforementioned classifiers.

1) Phishing Websites Dataset:[7] This dataset has 11,055 datapoints with 6,157 legitimate URLs and 4,898 phishing URLs. And it contains 30 features subdivided into three categories:
   a) Features based on the domain and subdomains

b) Features derived from the other parts of the URL
c) Features derived from the webpage HTML and JavaScript

2) Datasets for phishing websites detection:[8] This dataset consists of 111 features, of which 97 are based on the URL. For the phishing websites, the list was extracted from the PhishTank registry which are verified by multiple users. And for sets of legitimate websites, the publicly available and community labeled lists are utilized.[9]

When the server receives the URL, before it can be tested by any of the Machine Learning models, it are checked against a whitelist. The whitelist is extracted from The Majestic Million dataset[10] which maintains a list of top 1 million domains on the internet. The whitelist helps in reducing the processing and network overhead of checking the most popular sites which would be visited by an average user on a day to day basis. The experiments sections further elaborates on the need for doing this.

The machine learning models are already trained with both datasets, are precomputed and stored on the server. If the given domain is not part of the whitelist, the URL is then passed on to these models. Whenever the server receives a URL to be tested, it extracts all the features from the given URL. Many of the features are extracted through string parsing and the rest of them require use of external APIs and libraries. For example, PageRank of a given domain is fetched using Open PageRank API.[11]

After extracting the relevant features for each dataset, they are passed on to the classification models and their ensembles. And the result is sent back based on the outputs of the various models. One advantage of using a server based approach is essentially, results for different URLs from different users can be cached. This can save recurring computation for the same URL across users and decreases response times on repeated requests.

IV. Experiments and Results

Experiments in this report mainly concern 2 things - Effectiveness and Efficiency of Phishing detection. Due to the nature of the problem, the focus was more on effectiveness since there were a lot of parameters to tune. That being said, efficiency concerns are legitimate and were explored as well.

One of the important characteristics of a machine learning model is the data-set that is used to train it. The nature of the data-set greatly impacts the effectiveness of the result. Initially, Experiments were done on various data-sets that were available across the web using simple classification methods and the accuracy results were compared to the filter out the ineffective collections.

The decision to use two data-sets as a part of the final application was based on the observation that "Phishing Websites Dataset"[7] had a lot of non-url based features that which worked very well based off of the content of the website. In contrast, "Datasets for Phishing websites detection"[8] dataset contains 97 features that depended only on the URL structure, which are effective in detecting phishing websites. The inclusion of both these data-sets, with the result from their combined models, effectively acts as a way for the result to have legitimate checks and balances as well. If both the models predict that a website might be a phishing website, there is a high probability that it is true. Whenever there is a partial agreement, it is prudent to let the user know and be the final-arbiter of this conflict. This exploration allowed the feature of "Caution" vs "Alert" notifications to come up organically.

The models were built by splitting a portion of the input as training and test data, using a Decision Tree classifier and cross-validated into 10 splits. It was verified manually by using it as an extension in regular day-to-day browsing, to flag obvious errors. For the model from the dataset "Datasets for phishing websites detection",[8] it was noticed that the accuracy of the model was high with the test data, but performed rather poorly with random samples from the real world. This lead to the hypothesis that the model was over-fitting the data-set. This lead to the introduction of the ensembles for this model in particular. The combination of additional classifiers were tried out (Decision Tree, k-Nearest Neighbors and Random Forest) and verified with the same methods, until a minimum accuracy of 80% was achieved.

Tuning individual classifiers is also detrimental to the overall accuracy measurement of a model. For the "Datasets for phishing websites detection"[8] model, the classification-ensemble contained a K-Nearest Neighbor classifier. The number of neighbors for the classification was increased to the effect, that it over-fit the data. This was fruitful, since the ensemble of different classifiers helped reduce overall variance of the model.

The above experiments were predominantly effectiveness-based approaches. Although using a combination of two models helped improve the overall goal of detecting phishing websites, it came at the cost of the delay in prediction. Steps were taken to parallelize the flow of aggregating the predictions from the two models. That being said, there is no denying the fact that using a classifier to predict outcomes is inherently an expensive process. In a resource constrained environment, it must be called upon only when truly necessary. And once it's called upon, it is prudent for its results to be reused, to save the cost of classifying again. To this effect, a "Whitelist" of frequently used websites was added to skip prediction entirely, for overtly obvious websites. Since the intended application of this classification is in browser extensions, the fact that users will browse popular websites frequently is not an entirely unfair assumption. Caching the results on the server also goes well with this approach. This rule-based counterpart for the Machine Learning classifier helps improve the overall efficiency of the server.

The implementation of the report can be found in this repository, which contains the source-code as well as steps to get it running on a local machine - https://github.com/gnithin/phish-hook

TABLE I
PERFORMANCE RESULTS OF THE MODELS

Model	Accuracy	Precision	Recall	F1 Score
Ensemble Model[8]	90.68	92.69	89.20	90.92
Decision tree Model[7]	80.37	84.03	80.82	82.40

V. CONCLUSION AND FUTURE WORK

Phishing is one of the most widespread security attacks on the internet of this age. As the attacks keep getting more and more sophisticated, the solutions to prevent them need to keep with them. The traditional methods to detect attacks need to be combined with the growing fields of machine learning and artificial intelligence. It is apparent from various experiments and observations that the ideal solution to such issues, if any, is to combine best of all approaches and make them work with each other.

For the enhancement of this implementation, there seems to be a potential to improve the machine learning models as well as the features that are being used for classification. The extension and the server can be updated to also take in user feedback regarding a misclassification of any site. And the feedback would be considered by the classification models to improve their performance. Although, it might introduce a fresh set of challenges such as abuse of the feedback system to pass off an illegitimate site as safe (adversarial attacks). Potential solution to this problem would be to have a threshold on number of feedbacks until which the system does not consider them for the particular website.

The solution proposed and implemented in this project can be a genesis to a series of tools and products which strive to solve the issue of phishing attacks on the internet. The existing rule-based solutions which use HTML properties of the webpage can be incorporated into the machine learning models for potential improvement in performance. The existing REST APIs and the Python interfaces can also be reused to monitor phishing in other contexts. There can be an Android application which monitors the URLs visited by the user and notifies them if any of them have a potential to be a threat. And the same use case can be applied to other platforms, operating systems and at different abstraction levels like network or system.

VI. CONTRIBUTIONS

This project has been a collaborative effort. Most of the tasks were done together. These tasks include - Scouting for data-sets, extraction of features and creating Machine Learning models, tuning and testing the extension, and programming the Chrome extension.

In some of the tasks, one person shouldered more responsibility while the other followed. Tanmay took the lead in writing up the report in Latex, while Nithin worked on setting up the REST endpoints in Python.

REFERENCES

[1] Proofpoint, "Threat report: 2021 state of the phish report." [Online]. Available: https://www.proofpoint.com/us/resources/threat-reports/state-of-phish

[2] M. Moghimi, "Phishdetector - true phishing detection." [Online]. Available: https://chrome.google.com/webstore/detail/phishdetector-true-phishi/kgecldbalfgmgelepbblodfoogmjdgmj

[3] Cloudphish, "Cloudphish anti-phishing extension." [Online]. Available: https://chrome.google.com/webstore/detail/cloudphish-anti-phishing/fcahokjdmffdhglnlhgbceafccfdfjkd?hl=en

[4] B. Arca, "Blue arca anti-phishing extension." [Online]. Available: https://chrome.google.com/webstore/detail/blue-arca-anti-phishing-e/nbladngkelnbhcinepiogponadjggkcd?hl=en

[5] Retruster, "Retruster phishing protection." [Online]. Available: https://chrome.google.com/webstore/detail/retruster-phishing-protec/akcpbmbdplmbhlpeglpbghnkcbhiapil?hl=en

[6] A. Zuelsdorf, "Phishing boat." [Online]. Available: https://chrome.google.com/webstore/detail/phishing-boat/ljaiihgfejfaggbjfildfnjdckomlfop?hl=en

[7] M. et al, "Phishing websites data set by uci." [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Phishing+Websites

[8] G. Vrbančič, "Datasets for phishing websites detection." [Online]. Available: http://dx.doi.org/10.17632/72ptz43s9v.1

[9] T. C. Lab, "Test lists." [Online]. Available: https://github.com/citizenlab/test-lists

[10] Majestic, "Majestic top 1 million websites." [Online]. Available: https://majestic.com/reports/majestic-million

[11] O. P. initiative, "Open pagerank api." [Online]. Available: https://www.domcop.com/openpagerank/