Fresh-Phish: A Framework for Auto-Detection of Phishing Websites

Hossein Shirazi, Kyle Haefner, Indrakshi Ray
Department of Computer Science
Colorado State University
Fort Collins, USA
Email: {shirazi, kyle.haefner , iray}@colostate.edu

***Abstract*—Denizens of the Internet are coming under a barrage of phishing attacks of increasing frequency and sophistication. Emails accompanied by authentic looking websites are ensnaring users who, unwittingly, hand over their credentials compromising both their privacy and security. Methods such as the blacklisting of these phishing websites become untenable and cannot keep pace with the explosion of fake sites. Detection of nefarious websites must become automated and be able to adapt to this ever evolving form of social engineering.**

We develop a framework, called "Fresh-Phish", for creating current machine learning data for phishing websites. Using 30 different website features that we query using python, we build a large labeled dataset and analyze several machine learning classifiers against this dataset to determine which is the most accurate. We analyze not just the accuracy of the technique, but also how long it takes to train the model.

***Keywords*-Phishing, Machine Learning, TensorFlow**

I. Introduction

Phishing, defined as, "the attempt to obtain sensitive information such as user-names, passwords, and credit card details, often for malicious reasons, by masquerading as a trustworthy entity in an electronic communication" [1], is a problem that is as old as the Internet itself. Trying to get unsuspecting users to give up their money, credentials or privacy is a particularly insidious form of social engineering that can have disastrous affects on people's lives. Often this type of attack arrives in the form of an email containing the first part of what Chaudhry et al. describe as the lure, the hook and the catch [2].

The lure is what entices the user to click on a link. It can be advertising a way to get easy money, obtain an illicit product, or a warning that a user's account has been compromised or blocked in some fashion. The hook is often a website that is designed to mimic a legitimate website of a reputable organization such as a bank or other financial institution. The hook is used to trick the user into entering and submitting their credentials such as user-name, password, credit card number, etc. The catch is when the user has submitted their private information and the malicious owner of the website collects and uses this information to exploit the user and his accounts.

Figure 1 shows the number of phishing attacks has been increased year over year for the last decade. Anti-Phishing Working Group (APWG) reported an alarming 250% increase from the last quarter of 2015 to the first quarter of 2016 [3].

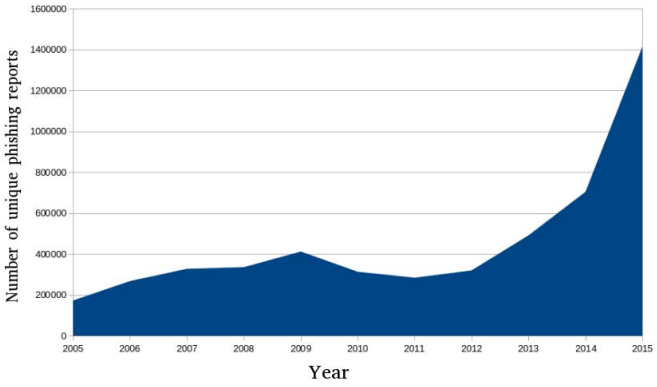

Figure 1: Graph of reported phishing incidents - First quarter of 2016

Not only have phishing attempts evolved and become more sophisticated, the motivation for implementing these attacks has changed as well. Attackers today are no longer curious hackers probing the security of systems: their primary goal has become financial gain. Figure 2 charts the fourth quarter of 2016 showing that 41% targeted industries are retail/services and 19% of them financial institutions. This wide diversity of targeted services, coupled with the trend of increasing attacks demonstrates that end-users are in more danger from more sources than ever before. [4]

Phishing is a growing multi-vector problem that has real and devastating consequences for users. It is a problem growing in sophistication, scope and reach. Automated detection techniques are critical to a safe and secure Internet. We believe that machine learning is an ideal method for providing this automated detection, however work to date leaves out one critical variable in this equation; we need an open and extensible framework capable of generating up-to-date data for researchers. We call this framework, Fresh-Phish.

There is no recent machine learning data that has been published on phishing websites. The data that does exist is several years out of date, a serious problem given the

978-1-5386-1562-1/17 $31.00 © 2017 IEEE
DOI 10.1109/IRI.2017.40

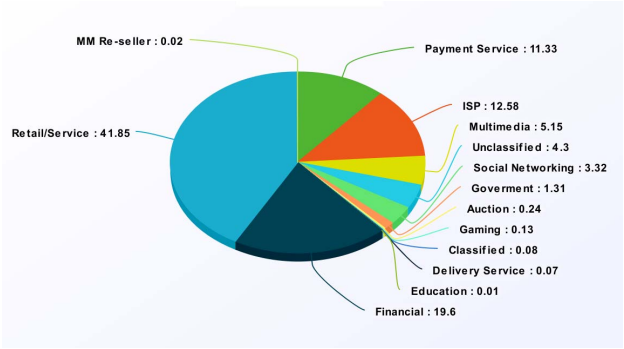

Figure 2: Graph of reported phishing incidents - Fourth quarter of 2016 [4]

dynamic nature of the Internet. There is also no published framework, that we are aware of, for gathering new data.

In this paper we introduce an open-source python-based framework called Fresh-Phish for generating up-to-date data of websites for training machine learning algorithms. The Fresh-Phish framework is intended to be an extensible building block that other researchers can modify, add, delete, or change what features are used to build datasets. We used our framework to crawl over 12,000 websites to generate a large labeled dataset with which we tested and analyzed several different machine learning techniques to accurately identify phishing websites.

The rest of the paper is organized as follows: In Section II we discuss several works that use automated techniques to identify phishing websites. In Section III , we layout how we implement our Fresh-Phish framework for calculating a phish rank on 30 website features originally defined by Mohammad et al. [5]. We show how we use our framework to build a up-to-date dataset with thousands of labeled examples. In Section IV we calculate which features are the most important in detecting phishing websites as well as examine various machine learning algorithms trained and tested on our dataset for accuracy and training time. In Section V we summarize how our open and published framework was built and how it can be successfully used to generate data for further research and also discuss future works with regards to the other features that we plan to explore and additional machine learning algorithms that we would like to apply for detecting phishing websites. In the Appendix we show how we define and calculate each website feature.

II. Related Works

Work to date on detecting phishing attacks largely follows a two pronged approach: detecting and filtering of phishing email, and detecting and filtering of the phishing websites that are linked in the email. Both of these approaches are necessary. Phishing email has become more sophisticated and targeted and can slip past filters. Additionally, there are several other vectors that are used by phishers that bypass emails such as malware attacks, session hijacking, search engine phishing, SMS, social networking and even online games! [6].

Basnet et al. employed a wide range of machine learning techniques including Support Vector Machines (SVM), Neural-Networks, self-organizing maps and K-Means to detect phishing emails [7]. They used features of emails only, not the websites that were linked in the email.

Miyamoto et al. [8] provide an overview of several different machine learning techniques, including SVM, Random Forests, Neural Networks, Naive Bayes and Bayesian Additive Regression Trees. They analyze how accurate each one is on a dataset developed by Z. Hong et al., called CANTINA [9]. Miyomoto et al. achieved a maximum accuracy of 91.34%.

The popular browser, Firefox, checks each website that you visit against reported phishing, unwanted software and malware lists. These lists are automatically downloaded and updated every 30 minutes or so when the "Phishing and Malware Protection" feature is enabled [10].

Finally we looked at Mohammad et al. [5] data published to the UCI database and their follow-up work that applies machine learning techniques to this data where they achieve an accuracy of 94.07%. They defined a good set of features and then created their dataset. [11]. While Mohammad et al. published just their dataset, we created an open-source framework which can measure all features defined by Mohammad et al. The framework can be used as a standard base in which to build up-to-date datasets as well as can be extended by other researchers to address the dynamic nature of phishing websites.

III. Methods

A. Creating a phishing dataset

The data based on features of websites on the Internet quickly become out of date and stale. To get an up-to-date analysis on the performance of our classifiers we built a framework for creating a new dataset for testing and it uses the same feature definitions that Mohammad et al. [5] used, but implemented in python. For determining Whois data, we used the API provided by Whoisxmlapi.com [12]. To create our dataset, we scanned the top 6000 sites in the Alexa database and 6000 online phishing sites obtained from phishtank.com. We made two assumptions here. First, all of the top 6000 websites on Alexa were legitimate sites. We believe this to be a valid assumption because of the ephemeral nature of phishing websites, they tend to pop in and out of existence (as is evidenced by the short domain registration times) to evade being blocked or tagged as phishing. The top 6000 sites ranked in Alexa must be popular, and have been around for a longer period of time to attain this ranking.

Second, we assumed that websites found on the phishtank.com were phishing websites. Phishtank incorporates a community of registered users who report sites as phishing. Each member is ranked by the community and builds a good reputation by correctly reporting if a website is phishing or not. Since it is a very well-known repository for phishing websites, we can trust its decision for labeling a website as a phishing one.

B. Implemented Features

Mohammad et al. [5] used 30 different features to create their dataset. In this work, we used their definitions to create our own dataset. These features can be categorized in five different categories:

- URL Based
- DNS Based
- External Statistics
- HTML Based
- JavaScript Based

1) URL Based: URL based features are based on some aspect of the URL of the website. Attackers try to use the URL to deceive users by obsufucating it in some fashion. For example, URLs that have an IP address, an 'at'' symbol (@), double slash, contain a prefix or suffix are all methods employed to disguise a URL. Other notable methods are the length of URL, whether the website has a sub-domain, uses a shortening service or uses a non-standard port.

2) DNS Based: DNS based features use information of the domain such as when the domain was first registered and how long the registration is valid.

3) External Statistics: External statistics based features use data gathered from places like Alexa's page rank, and if the site is present in Google's search index.

4) HTML Based: The HTML served by a website contains many valuable features used to determine if the site is phishing or not. Examples of these features include whether the website has a favicon, and if the images and JavaScript have the same source URL as the serving website. Other HTML based features are whether the site implements iFrames, how many links point outside the serving domain, etc.

5) JavaScript Based: JavaScript based features look for specific ways that JavaScript can be used to trick the end user. Some of these include things like using JavaScript to submit form data to email, mouse over techniques that hide URLs or prevent right clicks and pop-up windows.

The definitions of the features are described in Appendix.

C. Fresh-Phish dataset

Using our framework we were able to gather information about 6000 legitimate and 6000 phishing websites. We have published this dataset and made it publicly available on [13]. It includes values for 30 different features, assigned label and URL of each data item. Each feature was assigned a value based on if it was believed to be legitimate (-1), or phishy (1). For non-binary features like (age of domain or linksPointingToPage) we used a threshold. You can find detailed explanation of each in the Appendix.
Each example had a corresponding label: 1 for phishy, -1 for legitimate.

D. Classifiers

We implemented four classifiers using the Tfcontrib [14] library, and two classifiers using the scikit-learn library [15] to compare their accuracy on our dataset.

Using TensorFlow and TFcontrib we built a deep neural network (DNN) using the following built-in optimizations, Adadelta, Adagrad, and GradientDescent. For each optimization we varied the structure of the neural network by changing the number of hidden layers from one to three and varying the number of neurons in each hidden layer from 10 to 1000. We also used TensorFlow to implement a linear classifier.

Using scikit-learn we built an SVM using stratified K fold for validation and grid search with cost $C = [10^{-3}...10^{5}]$ and $\gamma = [10^{-3}...10^{5}]$. The best hyper-parameters found were: $C = 100$ and the kernel coefficient gamma: $\gamma = 0.316$ The best kernel was the Gaussian/RBF kernel. Finally we built a SVM using a linear kernel, again using stratified K fold for validation.

The code for the Fresh-Phish framework has been published on Github [16]

IV. Results and Discussion

A. Feature Elimination

To find the most important features in our dataset, we used the coefficients of a linear model as an external estimator that assigns weights to features. We next employed a recursive feature elimination (RFE) approach to reduce the set of features to the most relevant by creating an RFE object in scikit-learn and computing a cross-validated score whose accuracy is proportional to the number of correct classifications.

Figure 3 shows the trend of accuracy versus selected features. According to Figure 3, we recursively train the model and at each step we remove the weakest performing feature. Each feature is weighted according to when it was removed. The figure shows the accuracy as we add our top performing features into the model. Accuracy climbs until we reach the tenth most important feature, thereafter accuracy declines sharply and then becomes flat.

Based on what we have done, the following are the list of 10 most important features of Fresh-Phish dataset:

- URL length
- Using shortening service
- Prefix and suffix
- Uses non-standardstandard port

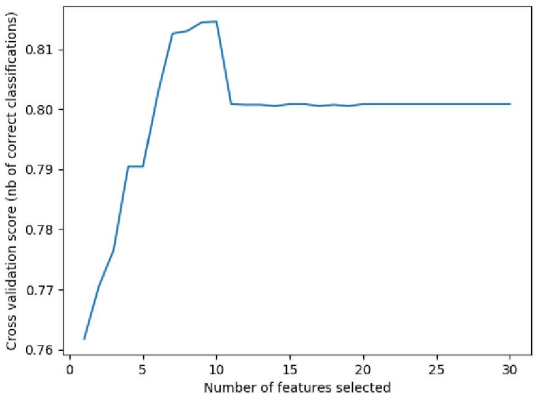

Figure 3: Feature removal vs. Accuracy

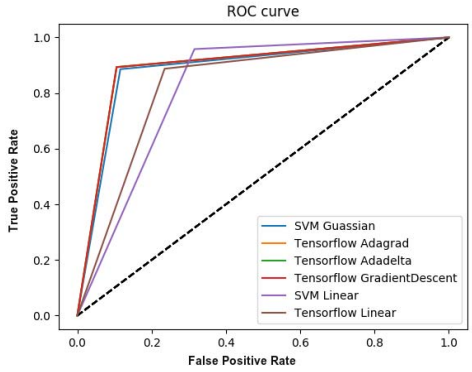

Figure 4: ROC Curve of implemented models

- HTTPS token
- Request URL
- Redirect page
- OnMouseOver
- PopUp widnow
- Google index

For details of features definition and implementation, please refer to Appendix.

B. Receiver Operating Characteristic of Classifier

For observing True-Positive (TP) and False-Positive (FP) rates in a classifier, a very common method is to plot Receiver Operating Characteristic (ROC) [17]. Figure 4 shows the ROC curve for the six models implemented. As seen in the figure, the TensorFlow neural network using the GradientDescent optimizer is the best model followed closely by SVM using the Gaussian kernel. The least effective classifiers were the two linear classifiers.

C. Accuracy of Classifier

We ran all of our trained classifiers against the data collected by the Fresh-Phish framework. We randomly selected 80% of Fresh-Phish dataset for training and reserved 20% for testing. We ran this experiment 10 times and used the average as the final results.

Table.I shows the Accuracy Under the Curve (AUC) for all six different classifier which we tested on our data set as well as the true-positive and true-negative accuracy of each website. With the exception of linear SVM and TensorFlow, all classifiers have an AUC around 90%. Notably, the results show that linear classifiers are not an ideal model for this data.

Accuracy of legitimate websites for different classifiers are around 90% except linear ones. Accuracy for phishing sites are around 89%. This symmetry shows that the classifiers were able to correctly label both true-positives and true-negatives with an acceptable rate.

An interesting observation here is: SVM linear has a very good result for detecting phishing websites even though it is not as good at detecting true-negative legitimate sites as other classifiers. It can predict phishing websites around 91% , which is an acceptable rate.

Table I: Data Accuracy

Classifier	**Accuracy**		
	AUC	TN	TP
TensorFlow Adagrad	0.8973	0.9027	0.8933
TensorFlow Adadelta	0.8970	0.9038	0.8916
TensorFlow GradientDescent	0.8972	0.9028	0.8928
TensorFlow Linear	0.8153	0.7676	0.8692
SVM Guassian	0.8932	0.8952	0.8921
SVM Linear	0.8250	0.7716	0.9177

Table.II shows the fitting and predicting time. True-Negative (TN) shows the percentage of legitimate websites which are correctly identified by the classifier while True-Positive (TP) is the percentage of phishing websites that have been correctly detected. It is obvious that TensorFlow consumed much more processing time than SVM for fitting. The prediction time for all of the classifiers is nominal. For all of our experiments we fit the classifier 10 times over a shuffled dataset and calculated the average time to train. For each of these trained classifiers we calculated the average prediction time. For this dataset the results show that the extra training required of neural networks over SVM was not worth the minuscule gains in accuracy.

V. Conclusion and Future works

Detecting phishing websites is an evolving game of cat and mouse. Publishers of phishing websites have developed increasingly sophisticated techniques and authentic looking sites to fool unsuspecting users. We developed the Fresh-Phish framework, as there is no open-source framework which measures features for any given website. Also, we

Table II: Training Time and Predicting Time

Classifier	Average Time	
	Fit Time (s)	Predict Time (s)
TensorFlow Adagrad	181.01	0.62
TensorFlow Adadelta	186.47	0.63
TensorFlow GradientDescent	178.31	0.52
TensorFlow Linear	139.48	0.13
SVM Guassian	7.13	0.38
SVM Linear	3.75	0.28

created an up-to-date data set which can be used by other researchers. We took 6000 clean websites and 6000 phishing websites and created our Fresh-Phish dataset subsequently we trained our classifier over this dataset.

We also analyzed a TensorFlow-based neural network, a TensorFlow-based linear classifier, an SVM with a Gaussian kernel and an SVM with a linear kernel against the Fresh-Phish data set. We found that the TensorFlow implementations took significantly longer to train while being only marginally more accurate than the SVM. We achieved an accuracy as high as 90% for the Fresh-Phish test data using a Gaussian kernel SVM.

In our future work we will improve our Fresh-Phish framework to increase its accuracy by determining features that distinguish between legitimate and phishing websites. Instead of relying on others' definitions for features [5] we will define our own. Additionally we will not use just binary values calculated by an arbitrary threshold, but will include ranges for such features where it makes logical sense, for example URL Length. Also, we will examine which features are no longer relevant. For example, the onMouseOver feature which attempts to mask a phishing URL in the browser status bar has become largely invalid as modern browsers no longer allow this to be exploited.

Next we will explore further correlations between phishing sites and hosting and DNS registration companies. We will also look at additional features that can be leveraged, such as Content Security Policies, certificate authorities, and TLS fingerprinting. Additionally, we will implement other machine learning techniques such as random forest classifiers for speed and accuracy and compare them to SVMs, and neural networks.

ACKNOWLEDGEMENTS

This work was supported in part by a grant from NSF under award number CNS 1650573 and funding from CableLabs, Furuno Electric Company, SecureNok, and Air Force Research Laboratory.

We would like to thank the WhoisXMLAPI website who supported us with their XML API to getting information of DNS records of each websites.

REFERENCES

[1] Wikipedia, "Phishing — Wikipedia, the free encyclopedia," 2016. [Online; accessed 21-October-2016].

[2] J. A. Chaudhry, S. A. Chaudhry, and R. G. Rittenhouse, "Phishing attacks and defenses," *International Journal of Security and Its Applications*, vol. 10, no. 1, pp. 247–256, 2016.

[3] Various, "Anti-phishing working group," 2016. [Online; accessed 04-April-2017].

[4] Various, "Anti-phishing working group," 2017. [Online; accessed 04-April-2017].

[5] R. M. Mohammad, F. Thabtah, and L. McCluskey, "An assessment of features related to phishing websites using an automated technique," in *Internet Technology And Secured Transactions, 2012 International Conference for*, pp. 492–497, IEEE, 2012.

[6] J. Hong, "The state of phishing attacks," *Communications of the ACM*, vol. 55, no. 1, pp. 74–81, 2012.

[7] R. Basnet, S. Mukkamala, and A. H. Sung, "Detection of phishing attacks: A machine learning approach," in *Soft Computing Applications in Industry*, pp. 373–383, Springer, 2008.

[8] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites," in *International Conference on Neural Information Processing*, pp. 539–546, Springer, 2008.

[9] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web*, pp. 639–648, ACM, 2007.

[10] Wikipedia, "Mozilla. phishing protection," 2017. [Online; accessed 04-April-2017].

[11] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Predicting phishing websites based on self-structuring neural network," *Neural Computing and Applications*, vol. 25, no. 2, pp. 443–458, 2014.

[12] Various, "Unified and consistent whois api and whois parser system," 2016. [Online: accessed 21-October-2016].

[13] I. R. Hossein Shirazi, Kyle Haefner, "Fresh-phish dataset: a dataset of phishing and legitimate website," 2016. [Online; accessed 05-April-2017].

[14] Various, "Tf-contrib learn library," 2016. [Online: accessed 21-October-2016].

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] I. R. Hossein Shirazi, Kyle Haefner, "Fresh-phish classifier: classifiers for predicting phishing and legitimate websites," 2016. [Online; accessed 05-April-2017].

[17] D. J. Hand, "Measuring classifier performance: acoherent alternative to the area under the roc curve," *Machine Learning*, vol. 77, no. 1, pp. 103–123, 2009.

APPENDIX

A. URL Based

1) Having 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. In a Python script, we checked that if the website URL is in the form of an IP, we will assume it as a phishing website otherwise it is a legitimate.

2) URL Length: To ensure accuracy of our study, we calculated the length of URLs in the data set 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.

3) Shortening Service: URL shortening is a method on the âĂIJWorld 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". If it used TinyURL, we will assume it as a phishing, otherwise, it is a legitimate website.

4) Having At Symbol: A URL that contains a "@" symbol is not trusted as the browser generally ignores everything proceeding the "@". If the URL contains the "@" sign we marked it as phishy.

5) Double Slash Redirecting: URLs that contain "//" are marked as phishy as the double slash is used to redirect users to another site. Phishing URLs emply this method to hide their real URL. An example is
http://www.colostate.edu//http://www.phishing.com.

6) Prefix Suffix: 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/

In our framework, we checked that website use a "-" in the name of URL or not. If it is used, we assume it as a phishing website.

7) Having SubDomain: 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.

We calculated number of dots in a URL. If it is more that 2 dots found, that will be phishing otherwise it is a legitimate website.

8) sslFinalState: This feature was not defined by Mohammad et al, nor did we get a response from them in email. We did not implement this feature and marked it as neutral on all examples.

9) port: Most legitimate website use ports 80 for unencrypted traffic and port 443 for encrypted traffic. Sites that use other ports are marked as phishy.

10) abnormalURL: This features used Whois data. If the Identity field in Whois does not match the domain in the URL this is marked as phishy.

B. DNS Based

1) domainRegistrationLength: This feature uses data from Whois. If the field "updatedDate" phishy domains are typically not registered and paid for for multiple years. If the updated date is less than half of a year the site is marked as phishy.

2) httpsToken: Phishing URLs will often try to make it look like the URL uses HTTPS. They will include HTTPS has part of the URL, for example, http://https-colostate.edu. These URLs are marked as phishy.

3) ageOfDomain: This feature can be extracted from WHOIS database. 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. Rule: IF

$$Feature_Value = \begin{cases} Legitimate, \\ \text{Age of Domain >Two Years.} \\ Suspicious, \\ \text{Age of Domain >One Year} \\ \text{AND} \\ \text{Age of Domain <Two Years.} \\ Phishing, \\ \text{Otherwise.} \end{cases} \quad (1)$$

We implemented a Phython script which retrieves DNS information from www.whoisxmlapi.com and then we calculated the value of ageOfDomain.

4) dnsRecord: This feature can be extracted from Whois database [11]. For phishing sites, either the claimed identity in not recognized by Whois database or the record of the host-name is not founded. If the DNS record is empty or not

found then the website is classified as "phishing", otherwise it is s classified as legitimate.

With implementing a python script which get DNS information from www.whoisxmlapi.com, we check is that domain record is empty or not.

C. External Statistics

1) pageRank: This feature looks at if the site is ranked in the Alexa database. If the site is not ranked or has no traffic then the site is marked as phishy.

2) googleIndex: 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. Usually, phishing web pages are merely accessible for a short period and as a result, many phishing web pages may not be found on the Google index. To finding Google Index of each site, we send a request to Google website then search website inside the result. If a website is indexed by Google, we mark it as legitimate, otherwise, we mark it as phishy.

3) statisticalReport: This feature uses data from other Phishing site trackers such as Phishtank.com. We did not implement this feature and set it to neutral.

D. HTML Based

1) 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.

For this attribute, we checked HTML code of each website and found where Favicon is loading from. If it is loaded from a foreign domain, we assumed that website as a phishing.

2) requestURL: 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. We implemented a Python script which look at all of address and mark them as domain-inside or domain-outside. If more than half of addresses are domain-outside, we will mark the site as phishing otherwise it is a legitimate one.

3) urlOfAnchor: This feature looks at the links in the website. If the links in the website point a a domain different from the domain of the website more that 50% of the time, then the site is marked as phishy.

4) linksInTags: This feature looks at the domain in the tags of the header such as <SCRIPT>, <META>, and <LINK> tags. If more than 50% of these tags point to a domain different from that of the site, the site is marked as phishy.

5) Submit Form Handler: This feature examines the action of the submit form on the page. If the action is, "None", "blank", or "about:blank", then the site is marked as phishy. Legitimate sites will point to a URL.

6) redirect: If the site uses the HTML 301 redirect in the header, then the site is marked as phishy.

7) iFrame: HTML used the <IFRAME> tag to display another page inside of the current page. This feature looks at if there is an <IFRAME> tag in the page and its border is set to transparent. If these two things are present mark the website as phishy.

8) linksPointingToPage: This feature looks at how many links from other websites are pointing the the target site. If there are no links to the target page, then it is marked as phishy. We did not implement this and defaulted it to neutral.

E. Javascript Based

1) submittingToEmail: This feature looks for a "mailto:" action in the submit form. If it exists then mark the site as phishy.

2) onMouseOver: This method looks for the on mouse over re-writing of links in the status bar. This type of ruse has become less effective as browsers usually ignore this. We used the python library Dryscrape to run a headless instance of webkit. This allows us to run and evaluate Javascript linked or embedded in the page. If the window.status Javascript call exists in congunction with onMouseOver then this site is marked as phishy.

3) rightClick: This feature looks for Javascript code that disables the right click action on a web page. This is meant to deter users from looking at the HTML source code for the site. It looks specifically for "event.button==2" in the Javascript. If that is present the site is marked as phishy.

4) popUpWindow: This method uses dryscrape which imlements webkit and can scrape a web page for Javascript has well as HTML. Javascript has alert,confirm,prompt,window.open methods if any of these are found then the site is marked as phishy.