Improving Auto-Detection of Phishing Websites using Fresh-Phish Framework

Hossein Shirazi, Kyle Haefner, Indrakshi Ray
Colorado State University, USA

{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 improved a framework that we previously implemented called "Fresh-Phish", for creating current machine learning data for phishing websites. The improved framework uses a total of 28 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. This modified framework improves the accuracy of modeling those features by using integer rather than binary values where possible. We analyze not just the accuracy of the technique, but also how long it takes to train the model.

Keywords: Cyber-Security, Phishing, Machine Learning, TensorFlow, SVM

INTRODUCTION

The Internet has ushered in a new evolution of electronic deception called phishing, that involves the one-two punch of web and email that is very difficult for users to detect. In fact, according to Alsharnouby et al. only 53% of users successfully detect phishing websites (Alsharnouby et al., 2015).

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" (Wikipedia, 2016), 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 effects 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* (Chaudhry, Chaudhry, & Rittenhouse, 2016).

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 increasing 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 (APWG, 2016).

Not only have phishing attempts evolved and become more sophisticated, the motivation for implementing these attacks has changed as well. Attackers today have moved beyond simply probing the security of systems; now their primary goal has become financial gain. This commercialization of phishing is charted in Figure 2 showing the

fourth quarter of 2016 where 41% of 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.

Phishing is a growing multi-vector problem that has real and devastating consequences for users. It is also a problem growing in sophistication, scope and reach. Automated detection techniques are critical to a safe and secure Internet. We use machine learning algorithms because they have been proven to have the capability to discover complex correlations among different data items of similar nature, 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 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 5,000 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 the related work section, we discuss several works that use automated techniques to identify phishing websites. In the methods section, we layout how we implement our Fresh-Phish framework for calculating a phish rank on 28 website features originally defined by Mohammad et al. (Mohammad, Thabtah, & McCluskey, 2012). We show how we use our framework to build an up-to-date dataset with thousands of labeled examples. In the results and discussion section 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 the conclusion and future work section we summarize how our open and published framework was built and how it can be successfully used to generate data for further research and discuss future work with regards to the other features that we plan to explore. Next, we look at additional machine learning algorithms that we would like to apply for detecting phishing websites. Finally, we compare the use of binary values for features versus using integer based values for features such as the length of a URL.

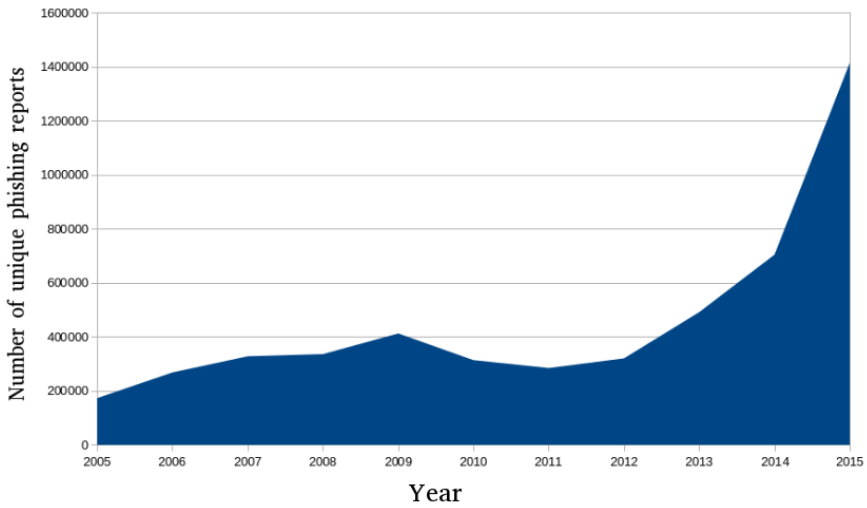

FIGURE 1 GRAPH OF REPORTED PHISHING INCIDENTS - FIRST QUARTER OF 2016 (APWG, 2016).

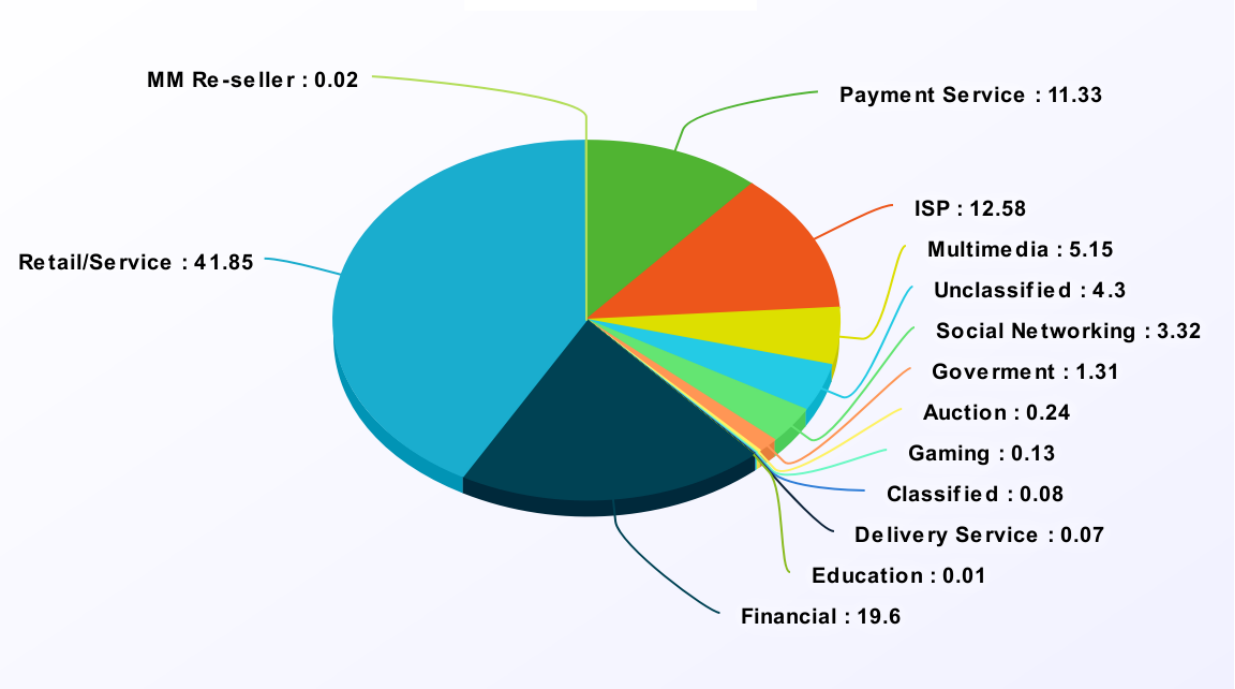

FIGURE 2 GRAPH OF REPORTED PHISHING INCIDENTS - FOURTH QUARTER OF 2016 (APWG, 2016A).

RELATED WORK

Work to date on detecting phishing attacks largely follows a two-pronged approach: detecting and filtering. This 'detect and filter' approach has increasingly become insufficient as attacks have become more complex and arrive from multiple sources. For example, phishing email has become more sophisticated and targeted. Often referred to as 'spear phishing' this type of attack can slip past statistical based filtering techniques. Additionally, there are several other vectors that are used by phishers that bypass email such as malware attacks, session hijacking, and search engine phishing, SMS, social networking and even online games! (Hong, 2012).

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 (Basnet, Mukkamala, & Sung, 2008). They used features of emails only, not the websites that were linked in the email.

Miyamoto et al. (Miyamoto, Hazeyama, & Kadobayashi, 2008) 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 (Zhang, Hong, & Cranor, 2007). Miyomoto et al. achieved a maximum accuracy of 91.34%.

Qian Cui et al. (Q. Cui, 2017) tried to find similarities between different attacks for 10 months study with monitoring more than 19000 websites. They found that 90% of attacks have similar DOM structure and over 90% of these attacks were actually replicas or variations of other attacks in the database. Zhang et al (Y. Zhang, 2017) created a dataset to make use of well-known TF-IDF (short term for Term Frequency–Inverse Document Frequency) algorithm to find out 5 most important words in each webpage. The 5 words are concatenated to form a lexical signature and then queried the lexical signature in Google search engine. If the website appears in the very first results of the query, then it is considered genuine. This approach provides an accuracy of 94-97%.

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 (Mohammad, Thabtah, & McCluskey, 2014).

Finally, we looked at Mohammad et al. (Mohammad et al., 2012) 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 (Mohammad et al., 2014).

While Mohammad et al. published just their dataset, we created a 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.

METHODS

Fresh-Phish Framework

The Fresh-Phish framework includes two major modules and each one has a series of Python classes: *Evaluation Module* which is responsible for creating the dataset itself by measuring different features within the website and *Experiment Module* which is responsible to calculating the phishing detection accuracy for the data in the previous step.

- *Evaluation Module*: This module, the first step of the framework, is used to prepare an updated dataset by calculating the set of defined features for each website. In this step, the module is fed a list of all websites to be measured, the features to be measured and assigns a label to each website. The module reads in a CSV file that includes the websites' URL, then passes this URL to a set of feature classes that parse various characteristics of the website into categories. For each category, we implemented a python class that evaluates all of features in that category to determine a phish score per feature. After evaluating all of features and calculating a phish score, we add the score label to each data item -1 for clean and 1 for phishy. This process will continue for all of websites in the CSV file.
- *Experiment Module:* After the dataset is created, the framework uses this module to conduct the experiments and calculate accuracy and other results. This module loads feature values and labels and runs a *K-Fold validation* approach to estimate the accuracy of each classifier. This process consists of training and testing steps. Finally, the module calculates the results from each classifier and reports them in a result file.

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. (Mohammad et al., 2012) used, but implemented in python. To create our dataset, we scanned the top 2500 sites in the Alexa database (Alexa, 2017) and 2500 online phishing sites obtained from PhishTank.com (PhishTank, 2017). We made two assumptions here. First, all the top 2500 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 2500 sites ranked in Alexa must be popular, and have been around for a longer period to attain this ranking.

Second, we assumed that websites found on the PhishTank.com were phishing websites. PhishTank.com 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.

Implemented Features

Mohammad et al. (Mohammad et al., 2012) used 28 different features to create their dataset using a three states definition for each feature as: phishing, neutral and legitimate. Limiting definition of each feature to a three-state variable removes some useful information for classifiers and might affect the accuracy. In this work, we used both three-state feature definition and original values of each feature to test its effect on the accuracy of the dataset.

The features can be categorized in five different categories:

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

In the following, we will explain each feature in the appropriate category.

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 "World Wide Web" in which a URL may be made considerably smaller in length and still lead to the required webpage. This is accomplished by means of an "HTTP Redirect" on a domain name that is short, which links to the webpage that has a long URL. For example, the URL "http://portal.hud.ac.uk/" can be shortened to "bit.ly/19DXSk4". 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 phishing.
5. **Double Slash Redirecting**: URLs that contain "//" are marked as phishing as the double slash is used to redirect users to another site. Phishing URLs employ 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 than 2 dots found, that will be phishing otherwise it is a legitimate website
8. **Unusual Port**: Most legitimate website use ports 80 for unencrypted traffic and port 443 for encrypted traffic. Sites that use other ports are marked as phishing.
9. **Abnormal URL**: This feature used Whois data. If the Identity field in Whois does not match the domain in the URL this is marked as phishing.

DNS Based

1. **Domain Registration Length**: This feature uses data from Whois. If the field "updatedDate" phishing domains are typically not registered and paid for multiple years. If the updated date is less than half of a year, the site is marked as phishing.
2. **HTTPS Token**: 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 phishing.
3. **Age of Domain**: 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 the age of domain is greater than 6 months, we will assume it as legitimate otherwise we will assume it as phishing.

4. **DNS Record**: This feature can be extracted from Whois database. 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 will classify as legitimate. With implementing a python script which get DNS information from www.whoisxmlapi.com, we check that domain if the domain record is empty or not.

External Statistics

1. **Page Rank**: 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 phishing.
2. **Google Index**: This feature examines whether a website is in Google's index or not. When a site is indexed by Google, it is displayed on search results. 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 phishing.
3. **Statistical Report**: This feature uses data from other Phishing site trackers such as PhishTank.com. We did not implement this feature and set it to neutral.

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 the 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. **Request URL**: This feature 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. **URL of Anchor**: This feature looks at the links in the website. If the links in the website point a domain different from the domain of the website more that 50% of the time, then the site is marked as phishing.
4. **Links in Tags**: 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 phishing.
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 phishing. Legitimate sites will point to a URL.
6. **Redirect Page**: If the site uses the HTML 301 redirect in the header, then the site is marked as phishing.
7. **Using 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 phishing.
8. **Links Pointing to Page**: This feature looks at how many links from other websites are pointing the target site. If there are no links to the target page, then it is marked as phishing. We did not implement this and defaulted it to neutral.

JavaScript Based

1. **Submitting to Email**: This feature looks for a "mailto:" action in the submit form. If it exists then mark the site as phishing.
2. **On Mouse Over**: 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 web-kit. This allows us to run and evaluate JavaScript linked or embedded in the

page. If the window.status JavaScript call exists in conjunction with onMouseOver then this site is marked as phishing.

3. **Right Click**: 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 phishing.
4. **Pop-up Window**: This method uses *Dryscrape* which implements web-kit 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 phishing.

Machine Learning Classifiers

Using TensorFlow and TFcontrib (Abadi et al., 2016) we built a Deep Neural Network (DNN) using the following built-in optimizations; vanilla GradientDescent, Adagrad, Adadelta. The vanilla gradient descent tries to minimize the cost function by calculating the gradients over the entire dataset. The Adagrad algorithm adapts the learning rate dynamically based on the frequency of parameters. It affects larger changes to the learning rate given infrequent parameters and smaller changes for frequent parameters. We used this algorithm because it lends itself to sparse data such as is constructed using our framework (Duchi, 2011). The Adadelta algorithm is a refinement of the Adagrad algorithm used to dynamically adapt the learning rate by using the decaying average of all previous squared gradients. (Zeiler, 2012). 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.

The SVM is a classifier based on finding a separating hyperplane in the feature space between two classes in such a way that the distance between the hyperplane and the closest data points of each class is maximized. Also, SVMs are well known for their generalization ability (Buczak, 2016), (Vapnik, 2013). We used this classifier two different kernels: *Linear* and *Gaussian*. In using SVM with a Gaussian kernel, we have to specify two important parameters: *cost* and *gamma*.

The *gamma* parameter affects how much influence a single example has; a low gamma imbues a Guassian radial bias with a large variance, while a high *gamma* makes the variance small and decreases the influence of the support vector. The *cost* parameter defines the penalty for misclassified examples. A low *cost* provides a softer or smother margin for the decision surface, while a high *cost* tries to classify all training examples correctly leading to hard margin. (Pedregosa, 2011).

To assign the *cost* and *gamma* parameters, we employed a grid search ranging from 10^{-3} to 10^{5} and the *gamma* from 10^{-3} to 10^{5}. The best hyper-parameters found were: *cost*=100 and *gamma* = 0.316.

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 (Hossein Shirazi, 2016a).

RESULTS AND DISCUSSION

Fresh-Phish dataset

Using our framework, we were able to measure features of about 2500 legitimate and 2500 phishing websites. We have published this dataset and made it publicly available on (Hossein Shirazi, 2016b). It includes values for 28 different features, assigned a label and URL for each data item.

We employed two different approaches to measure each feature. In our initial approach, we used a binary definition for features to create the dataset. We assigned a value based on if it was believed to be legitimate (+1), or phishing (-1). For non-binary features like (*Age of Domain* or *Links Pointing to Page*) we used a threshold, based on Mohammed et al feature definitions, to convert it to a binary value. In our second and improved approach, we created another dataset with original values of each feature without using thresholds to convert non-binary values to binary ones. For example, we use an integer length for the URL in the model. We found that using these integer values improved the model's accuracy and further discretization of the data was not needed.

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

(Granitto, 2006) 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. Per Figure 3, we recursively train the model and at each step we remove the weakest performing feature. Each feature is weighted by the fit estimator, in this case we employed a support vector machine with a linear kernel, and the least important features are removed. This procedure was recursively repeated until half of the feature set was removed. The figure shows the accuracy as we retain 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 is the list of the 10 most important features of Fresh-Phish dataset:

- URL length
- Using shortening service
- Prefix and suffix
- Uses non-standard port
- HTTPS token
- Request URL
- Redirect page
- OnMouseOver
- PopUp widnow
- Google index

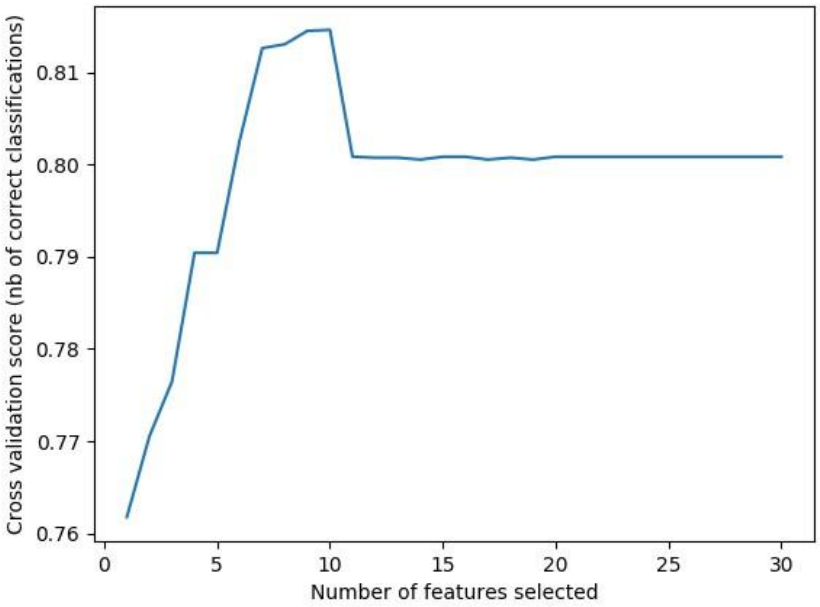

FIGURE 3 FEATURE REMOVAL VS. ACCURACY

Accuracy of Classifiers

We implemented four classifiers using the Tfcontrib (Abadi et al., 2016) library, and two classifiers using the scikit-learn library (Pedregosa et al., 2011) to compare their accuracy on our datasets.

Figure 4 and Figure 5 shows the Receiver Operating Characteristic (ROC) curve for binary and non-binary dataset respectively. Table 1 shows the Area Under Curve (AUC) for different classifiers on both binary and non-binary dataset. The AUC for binary dataset is around 90% except where linear classifiers were used. The AUC for non-binary classifier is around 92% for all classifier except for the linear. The best AUC for binary and non-binary datasets belongs to SVM-Gaussian with AUC of 0.930 and 0.961 for binary and non-binary datasets respectively.

This symmetry shows that the classifiers correctly labeled both true-positives and true-negatives with an acceptable rate. Also, it should be added that classifiers on the non-binary dataset work better that binary dataset so it means using non-binary features helped classifier to achieve better accuracy.

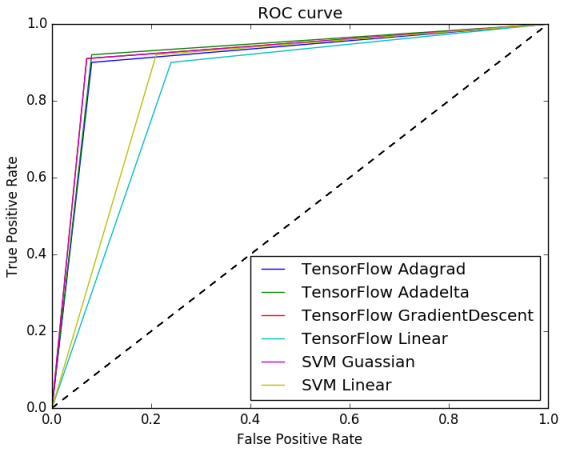

FIGURE 4 ROC CURVE OF IMPLEMENTED MODELS, BINARY DATASET

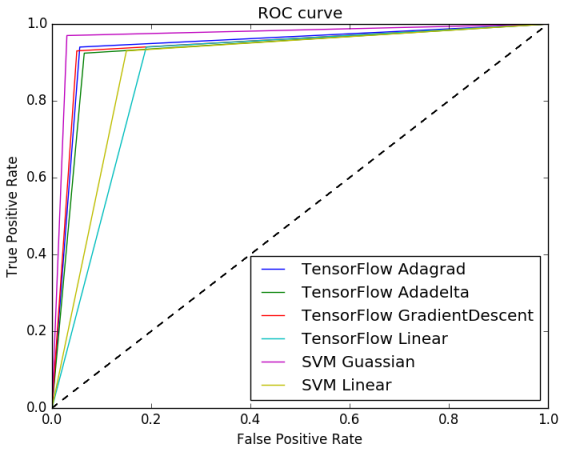

Figure 5 ROC CURVE OF IMPLEMENTED MODELS, NON-BINARY DATASET

TABLE 1 DATA ACCURACY

Classifier	AUC	
	Binary	Non-Binary
TensorFlow Adagrad	0.901	0.921
TensorFlow Adadelta	0.901	0.923
TensorFlow GradientDescent	0.904	0.922
TensorFlow Linear	0.765	0.821
SVM Guassian	0.930	0.961
SVM Linear	0.791	0.804

Training and Testing Time

Table 2 shows the fitting and predicting time for binary dataset. 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 the binary dataset, the results show that the extra training required of neural networks over SVM was not worth the minuscule gains in accuracy.

TABLE 2 TRAINING TIME AND PREDICTING TIME

Classifier	Time(s)	
	Training	Testing
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

CONCLUSION AND FUTURE WORK

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 created an up-to-date data set which can be used by other researchers. We took 2500 clean websites and 2500 phishing websites and created our Fresh-Phish dataset subsequently we trained our classifier over this dataset. To study the effect of using binary versus non-binary values for defined features, we created two different datasets. The one with binary values using threshold to convert actual values of features to the binary values and one dataset that we used actual values without converting them to binary values.

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. Also, we found that classifiers work great on both datasets. We achieved and accuracy of 93% on binary dataset and 96% on non-binary dataset which is acceptable rate. Also, it shows that the idea of using non-binary values helps to improve the accuracy.

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.

Finally, we will look at the underlying HTML structure for features, specifically counts of tags, placement of tags, use of and counts of specific JavaScript functions, inline and included CSS, etc.

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

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... others. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

Alsharnouby, M., Alaca, F., & Chiasson, S. (2015). Why phishing still works: User strategies for combating phishing attacks. International Journal of Human-Computer Studies, 82, 69–82. https://doi.org/10.1016/j.ijhcs.2015.05.005

APWG (Anti-Phishing Working Group). (2016). Phishing Activity Trends Report 1st Quarter 2016. Retrieved from http://docs.apwg.org/reports/apwg_trends_report_q1_2016.pdf

Basnet, R. B., Mukkamala, S., & Sung, A. H. (2008). Detection of Phishing Attacks: A Machine Learning Approach. Soft Computing Applications in Industry, 226, 373-383.

Chaudhry, J. A., Chaudhry, S. A., & Rittenhouse, R. G. (2016). Phishing Attacks and Defenses. International Journal of Security and Its Applications, 10(1), 247–256.

Hong, J. (2012). The state of phishing attacks. Communications of the ACM, 55(1), 74–81.

Hossein Shirazi, I. R., Kyle Haefner. (2016a). Fresh-Phish classifier: classifiers for predicting phishing and legitimate websites. Retrieved from https://github.com/FreshPhish/framework

Hossein Shirazi, I. R., Kyle Haefner. (2016b). Fresh-Phish dataset: a dataset of phishing and legitimate website. Retrieved from https://github.com/FreshPhish/dataset

Miyamoto, D., Hazeyama, H., & Kadobayashi, Y. (2008). An Evaluation of Machine Learning-Based Methods for Detection of Phishing Sites. In Advances in Neuro-Information Processing (pp. 539–546). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-02490-0_66

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2012). 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.

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2014). Predicting phishing websites based on self-structuring neural network. Neural Computing and Applications, 25(2), 443–458.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

APWG. (2016a). Anti-Phishing Working Group. Retrieved from http://docs.apwg.org/reports/apwg_trends_report_q1_2016.pdf

WHOISXMLAPI. (2016). Unified And Consistent Whois API and Whois Parser System. Retrieved from https://www.whoisxmlapi.com/

Wikipedia. (2016). *Phishing — Wikipedia, The Free Encyclopedia*. Retrieved from https://en.wikipedia.org/wiki/Phishing#cite_note-APWG-10

Zhang, Y., Hong, J. I., & Cranor, L. F. (2007). Cantina: A Content-based Approach to Detecting Phishing Web Sites. In Proceedings of the 16th International Conference on World Wide Web (pp. 639–648). New York, NY, USA: ACM. https://doi.org/10.1145/1242572.1242659

Alexa. (2017). The top 500 sites on the web. Retrieved from https://www.alexa.com/topsites

Phishtank. (2017). out of the net, into the tank. Retrieved from https://www.phishtank.com/

Granitto, P. M., Furlanello, C., Biasioli, F., & Gasperi, F. (2006). Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products. Chemometrics and Intelligent Laboratory Systems, 83(2), 83-90.

Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), 1153-1176.

Vapnik, V. (2013). The nature of statistical learning theory. Springer science & business media.

Q. Cui, G.-V. Jourdan, G. V. Bochmann, R. Couturier, and I.-V. Onut, "Tracking Phishing Attacks Over Time," in Proceedings of the 26th International Conference on World Wide Web, WWW '17, (Republic and Canton of Geneva, Switzerland), pp. 667–676, International World Wide Web Conferences Steering Committee, 2017. 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.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.