Building Robust Phishing Detection System: an Empirical Analysis

Jehyun Lee*, Pingxiao Ye†, Ruofan Liu*, Dinil Mon Divakaran†, Mun Choon Chan*
*National University of Singapore
{leejh, chanmc}@comp.nus.edu.sg, liu.ruofan16@u.nus.edu
†Trustwave
{Pingxiao.Ye, Dinil.Divakaran}@trustwave.com

***Abstract*—To tackle phishing attacks, recent research works have resorted to the application of machine learning (ML) algorithms, yielding promising results. Often, a binary classification model is trained on labeled datasets of benign and phishing URLs (and contents) obtained via crawling. While phishing classifiers have high accuracy (precision and recall), they, however, are also prone to adversarial attacks wherein an adversary tries to evade the ML-based classifier by mimicking (feature values of) benign web pages. Based on this observation, in our work, we propose a simple approach to build a robust phishing page detection system. Our detection system, based on voting, employs multiple models, such that each model is trained by inserting (controlled) noises in a subset of randomly selected features from the full feature set. We conduct comprehensive experiments using real datasets, and based on a number of evasive strategies, evaluate the robustness of, both, the traditional *native* ML model and our proposed detection system. The results demonstrate that our proposed system, on one hand, performs close to the native model when there is no adversarial attack, and on the other hand, is more robust against evasion attacks than the native model.**

I. Introduction

Phishing has been around since the early times of world wide web and is a common technique used to deceive and drive victims to subsequent attacks, such as leakage of sensitive information and malware infection. Despite persistent efforts, users are increasingly affected by different forms of phishing attacks (a recent report shows a significant increase in phishing [1]), with the medium of deception expanding from traditional e-mail systems to SMS, chat applications and social networks. These systems are used to deliver hypertext links (URLs) of HTML pages to victims, with the hope of persuading them to access the link.

While security vendors and researchers have been developing different kinds of solutions over the past many years [32], the increased level of sophistication employed by attackers has made detection of phishing pages a challenging problem. Recently, researchers have proposed machine-learning (ML) based approaches for efficient and accurate detection of phishing pages [17], [44], [26], [53]. Generally, such works extract information (features) pertaining to the phishing pages as well as legitimate (benign) pages and build a binary classifier by training on large labeled datasets of phishing and benign URLs and pages.

However, ML-based phishing detection solutions are also prone to adversarial attacks. With the knowledge of the classifier, an attacker can craft web pages with feature values mimicking that of the benign web pages, and also actively test against the trained classifiers. In fact, an attacker can use publicly available services to know the result of their evasion techniques [49], [29], [16], and subsequently release the phishing pages in the wild. Recent works have demonstrated the effect of evasion attacks against ML-based phishing detectors in which selected features are modified [27], [45]. In particular, copying the values for specific features from benign pages is apparently sufficient to degrade the ML-based classifier's performance significantly.

In this context, it is important to come up with a robust classifier, whose performance[1] does not degrade under evasion attacks. Broadly, there are two approaches to build a robust classification model. One is to come up with more features which are potentially harder for an adversary to manipulate (e.g., see [48]). Another possibility is to reduce the impact of a few important features on the accuracy of the classifier [25] which are exploited by an adversary. In this work, we focus on the latter approach.

We propose a simple but effective training methodology to build a robust phishing detection system. The basic idea of our approach is to deal with the *skewed feature-importance* problem in classifiers (particularly in tree-based classifiers such as Random forest) by adding controlled noises to the training dataset. We achieve this by training multiple models, such that each model selects a subset of all features used for training, and inserts noises to these selected features in a controlled manner. We propose and study two meta-classification systems to make a decision based on the multiple classifiers — one system randomly picks and employs one of the generated models, whereas the other uses voting to decide the outcome of classification. Our proposed system (based on voting) achieves enhanced robustness against the evasion attempts that mimic the features of the benign examples, while still maintaining higher performance (in terms of precision and recall) in the

[1]As phishing detection classification has an imbalanced class problem, the important metrics for performance analysis are precision and recall.

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-63-0
https://dx.doi.org/10.14722/madweb.2020.23007
www.ndss-symposium.org

scenario where there is no evasion.

We summarise our contributions in the following:

- Based on openly available datasets collected over three months for the benign URLs and one and half months for the phishing URLs, we build traditional ML classification models based on various features that are effective in detecting phishing pages with high precision and recall. Subsequently, and more importantly, we conduct systematic studies demonstrating the degradation of performance due to evasion attacks. Specifically, our results show that the area under precision-recall curve (AUPRC) decreases from 0.991 to 0.694 under adversarial attacks.
- We exploit a simple statistical technique and propose phishing detection systems that are robust against evasion attacks.
- Through extensive experiments, we study the robustness, generality and efficacy of our proposal system.

The rest of the paper is organized as follows. In the next section, after providing the background on phishing detection classifiers, we demonstrate the evasion attack and put forth the threat model that we consider in this work. As a countermeasure to the threat, we propose new detection systems that enhance the robustness of existing models, in Section III. Subsequently, in Section IV, we evaluate the performance of the proposed systems under multiple evasion strategies. We discuss further on evasion strategy as well as on potential future direction in Section V. We conclude after briefing on related works in Section VI.

II. Motivation

Machine learning (including deep learning) has been increasingly employed for security purposes, such as anomaly detection [34], [35], malicious domain detection [8], etc. In this section, we provide the background on phishing classifiers, including the different features used for building ML-based classifiers. Subsequently, we illustrate and thereby motivate the possibility of evading phishing classifiers built on such widely used features. Finally, we present the threat model considered in this work.

A. Background

Previous studies using machine learning techniques have mainly focused on defining new features that represent the behavior, semantic, page structure, reputation and many other characteristics of a target web site. The existing classifiers and their features have shown their efficacy in classifying phishing pages by modeling the malicious and abnormal contents of phishing pages using quantitative metrics (e.g., [43], [44], [53]).

We categorize the phishing classifiers based on the source of the features used for building the models: i) Input sources are URLs, which can be extracted before the target website is actually accessed. ii) Input sources are both URLs as well as HTML contents, the latter requiring more time and space to collect during operation. URL strings and HTML contents are usually accessible for a deployed phishing page detection solution, in either host-based or network-based systems. Many network-based detection systems provide phishing detection services for HTTPS sessions through a secure channel, such as a web-proxy and VPN tunneling [18].

Classifiers with URL-based Features: Detecting phishing pages based only on URL-based features is appealing for multiple reasons. One, URLs can be obtained from network traffic or web-proxies before the actual (malicious) page is loaded. Two, processing URLs and classifying pages using URLs can be achieved in real-time and faster than models that require HTML pages. Rakesh *et al.* [39] explored statistical features on URL strings such as length of URL, suspicious symbol counts, character frequency distribution, number of target brands, similarity to English dictionary, etc. This is based on the assumption that phishing sites generally target well-known sites, thus they will attempt to deceive users by embedding targets into a URL path. Furthermore, the study assumes the lexical structure of legitimate URLs and phishing URLs are inherently different. Authors in [43] proposed a methodology to quantify the intra-relatedness of tokens on a URL string. They argue that the tokens in a legitimate URL are related to each other, meanwhile, those in the phishing URLs are not. The proposal measures the quantified degree of relationship between the tokens by comparing the results from search engines.

Classifiers with HTML content-based Features: With the increased sophistication of URLs for phishing pages, the detection based only on URLs has faced some limitations [30]. Inspecting HTML contents is considered an expensive but worthy approach to figure out the malicious behavior which manifests more in the page contents than in the URL. Cantina+ [17] incorporates URL features, HTML features and web-based features, for building a phishing detection classifier. Traditional features in URL, e.g., sensitive words, number of dots, @ symbol, etc. are reused and two new features in URLs — IP address and out-of-position TLD — are also proposed. As for HTML, the work mainly considers on characterising the login form and hyperlinks. External features include WHOIS records and search engine results. Later works such as [22] and [53] further expanded the feature set, based on the insight that phishing sites include hyperlinks that are pointing outside of the domain, hence it is useful to compute the internal/external link ratio and internal/external resource ratio. In another work, Samuel *et al.* [44] gave attention to the relationship between the URLs and HTML pages during the browsing process for each single URL access. For example, an interesting intuition they put forth was that the frequently appearing word tokens should be consistent in the starting, landing and intermediate redirection URLs, and also in HTML contents, for benign URL access.

We trained different classifiers based on fifty-one phishing detection features from multiple previous works [17], [44], [46], [51], [28]; the feature categories and examples are listed in Tables I. For avoiding ambiguity, from now on, we use a phishing page which has a URL and an HTML page as an entity of classification. Fig. 1 plots the precision-recall curves (PRC) achieved by the classifiers on test data. Precision is the fraction of correctly predicted pages of all those predicted as phishing; whereas, recall is the fraction of correctly predicted pages of all those that were actually phishing URLs. A

TABLE I: Feature set used for building a test phishing page detection model. The full list of features is given in Appendix.

Source	# of features	Examples
URL string	13	Length of URL, Number of tokens in URL, Length of domain name, Number of dots in URL, Suspicious tokens in URL, etc.
HTML content	38	Number of images, Depth of DOM structure, Ratio of internal resources, Length of HTML text, Number of unique file types, etc.
Total	51	

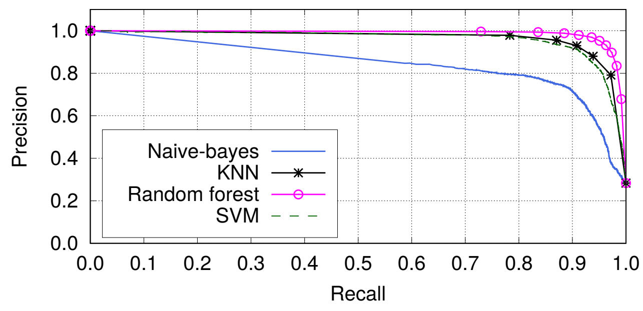


Fig. 1: Detection performance comparison of machine learning algorithms for phishing page detection

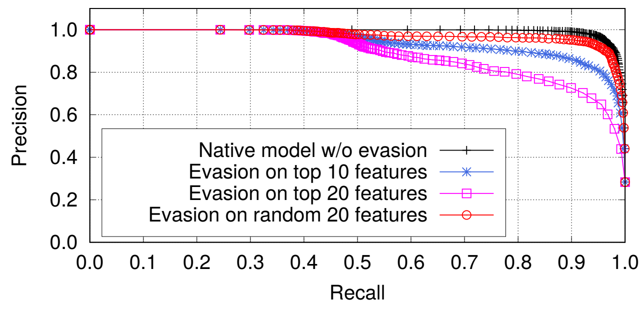


Fig. 2: Phishing page detection performance under evasion attack that mimics values from benign datasets for different selected features in 50% of the phishing pages

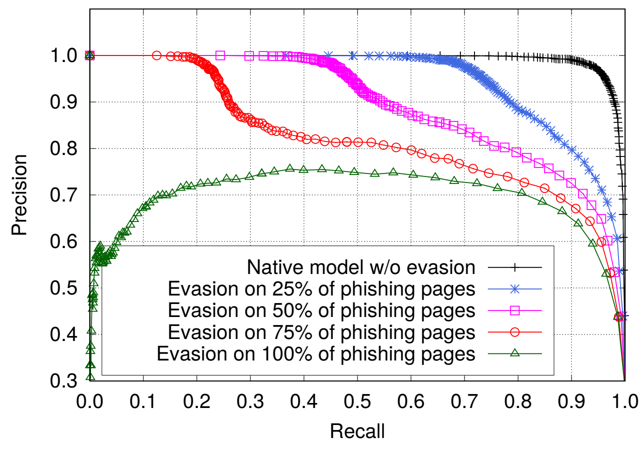


Fig. 3: Effect of evasion when only a certain percentage of phishing pages are modified for evasion purposes. Top 20 features were used for evasion attack.

classifier with recall 1.0 detects all phishing pages, but that is valuable only if the precision is high, otherwise, the number of false positives that an analyst would have to go through will be quite high. Therefore, from an operational perspective, having high precision is very important. From Fig. 1, we observe that Random forest classifier performs the best; henceforth, our discussions and evaluations will be based on Random forest classifier.

B. Evasion by mimicking benign values of targeted features

Many works on phishing classifiers are prone to adversarial attacks. On learning the feature set used for building the classification model deployed, an adversary can, with little effort, develop a phishing web page that easily evades the classifier. That is, an adversary can mislead the system to classify a malicious sample as a benign sample. In particular, an adversary can achieve this misclassification by exploiting the knowledge of the classification model. In a couple of taxonomies, this kind of attacks are categorized as *exploratory attacks* [4], [27]. Liang *et al.* [27] confirms that even a widely-used commercial phishing detection system can be bypassed by the exploratory attack. Once an adversary knows the features of the phishing classifier that are useful in differentiating the two classes (benign and phishing), the adversary can craft the phishing page in a way that, such features have appropriate values to evade the classifier.

To confirm the threat due to evasion attacks on a phishing classifier, we carried out a set of experiments on Random forest based phishing page classifier. Here we set a simple evasion attack assuming knowledge of the model — for the top important features of the classifier, the phishing pages copied the values corresponding to the benign web pages (which is also the goal of the attacker in any case, as a phishing page is supposed to appear very similar to the target benign web page). Fig. 2 plots the PRC of the traditional or *native* classifier when there is no evasion attack, and when there is an evasion attack mimicking top 10 and top 20 features for half of phishing pages in the test set[2]. As a consequence of the attack, 99.1% of AUPRC (area under PR curve) achieved under the naive condition falls to 94.0% and 89.1%, when top 10 and top 20 features, respectively, are evaded. AUPRC gaps between the different PRCs in the figure clearly confirm the efficacy of the evasion attacks and the degradation of the classifier. Fig. 3 plots evasion on top 20 features of the native model, when 25%, 50%, 75%, and 100% of phishing samples are modified for evasion. When the percentage of evading phishing pages increases to 75% and 100%, AUPRC drops further to, **81.9%** and **69.4%**, respectively. We can observe in Figure 2 that, selecting random features for evasion does not affect the performance significantly, and knowing the top discriminative features that help in classification is important for an adversary to craft effective evasive pages.

C. Threat model

We consider the ML-based phishing detection system as the primary victim of attacks. An adversary is the one behind a phishing attack; equally importantly, an adversary in our work attempts to evade the phishing detection systems so that the

[2]Details of the dataset and features for this scenario are given in Section IV.

phishing web page that has been set up is accessible to the end-user (also a victim).

Adversary capability: We assume that an adversary in our scenario has a general understanding of machine learning algorithms to evade the ML-based phishing detection system. An adversary may have control over the domain name and the content of a web server which serves malicious contents. To achieve this goal, adversaries need to have the ability to craft and register new domain names. An adversary may also have the capability to learn the important features used by a target detection system. There are two ways to achieve this: (i) *white-box attack*: the adversary could build a classifier using openly available datasets, such as Alexa top websites [2], PhishTank database [38], etc.; this is also the commonly used approach by defenders. Based on the classifier model built, the adversary can learn the important features of use. For example, training Random forest classifier, one can obtain an importance score of features used (which is also dependent on the correlation of features). There are also other ways of estimating feature ranks (or importance); see [13] for example. (ii) *black-box attack*: through repeated tests, an adversary could potentially guess the features used in the ML-model, as the adversary can also observe the result of his attack (if the phishing web page is accessed or not) [7], [27], [40].

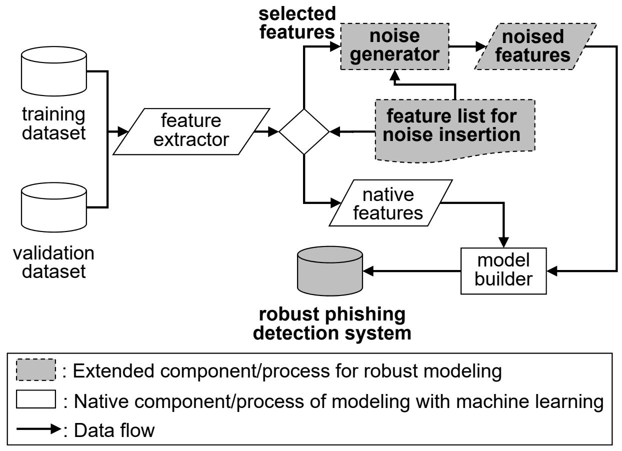
Scope and assumptions: We do not limit the features that can be exploited by an adversary, as long as the features are from URLs and HTML contents, both of which are under the control of the attacker. However, a phishing site cannot be exactly the same as a benign site for a practical reason; for example, a phishing site targeting a bank website, would need at least the URL to be different. We do not specify the method to drive a victim to the phishing web server; e.g., it could using spam email, social engineering, SMS, etc.

The *causative attacks* [5] that affect the training dataset or training process are not considered in our threat model. The causative attacks to a security application although aims to lower the system performance, its execution is a critical security breach of a confidential system; and this cannot be solved via robust modeling. Poisoning attacks including training data manipulation and injection as well as direct logic corruption are not considered here. Neither is the availability attack within our scope of work. The evasion attack is carried out only at the inference stage.

Lastly, we do not consider the techniques that require analysis of the web pages as rendered by the browsers as well as those that analyse the dynamic behavior of the pages, for example using visual comparisons [14], [31] and sandboxing approaches [19].

III. Robust phishing page detection with feature ranking control

In this section, we propose a methodology to build a phishing detection system that enhances the robustness of an existing phishing page classifier against evasion attacks. One straight-forward way to achieve this would be to train different classifiers with different sets of features (i.e., each classifier has some features removed); and then use an integrated model based on these classifiers, so as to hinder an adversary from learning the features used for classification. However,



selected features
noise generator
noised features
training dataset
feature extractor
feature list for noise insertion
native features
validation dataset
model builder
robust phishing detection system
: Extended component/process for robust modeling
: Native component/process of modeling with machine learning
: Data flow

Fig. 4: Process diagram of robust n-version phishing page modeling

the removal of features from a classifier is potentially risky because the removed features are no more considered (in that respective model) for the purpose of classification. Instead, we propose a different methodology, wherein multiple instances of classifiers are generated, but without changing the feature set and model-building algorithms. The multiple classifiers are created from the same feature set, by *manipulating* the actual values of *randomly* selected features. The detection system can then operate these different instances of the classifiers, which together are (possibly) more robust against the evasion attacks than one single classifier.

Figure 4 illustrates the modeling process for building a robust detection system using multiple classifiers. In the following, we describe the important steps involved in the process; i.e., (i) changing the feature ranking of a model using *noise insertion*, ii) creating multiple models having *diverse* feature rankings, and lastly, iii) building a robust phishing page detection system with the multiple models.

A. Noise insertion on feature values

The first step towards building a robust phishing detection model is to change the ranking of features. There are a few well-known techniques to achieve this [25]; one of them is to *inject* or *insert* noise to the values of a selected feature [12], [15]. Intuitively, adding noise decreases the feature weight within a classifier (such as a tree-based classifier), and consequently, other features (whose values have not been modified) gain importance in this new *noised* training dataset.

The noise generator, depicted in Figure 4, modifies the values (or data points) of a selected feature in the training dataset to arbitrary values. The algorithm for noise generation is guided by two principles: (i) the range of the original values must be retained because it is part of the feature characteristic; and (ii) change of value should not be very arbitrary, lest the correlation with another feature may be broken. Thus, we propose to shift a selected data point to random distance, but within a fixed range of the original value.

Algorithm 1 gives the pseudo-code for noise generation. $\mathbf{V}$ is the list of all data values for a particular feature f obtained

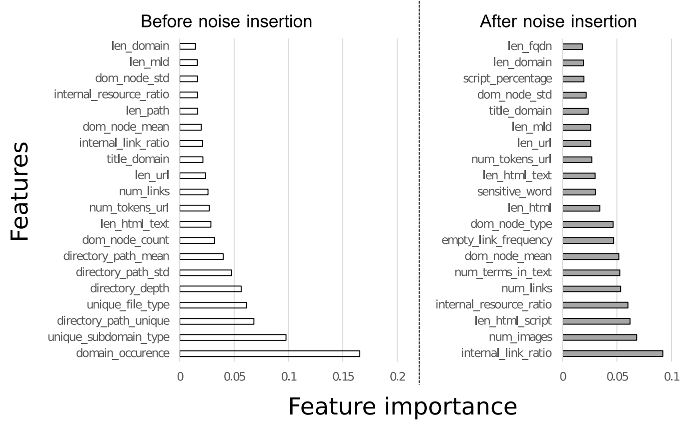


Fig. 5: Change on feature importance and ranking by noise insertion; top twenty features of the models before and after noise insertion.

from the training dataset (for simplicity, we do not use a subscript to indicate feature), and $\mathbf{W}$ is the output list with the *noised* values for the same feature f. Obviously, $\mathbf{V}$ and $\mathbf{W}$ are of the same length N. Noises are generated differently for different data types — binary, continuous real numbers, and discrete numbers. For binary values, they are flipped based on the Bernoulli distribution with probability p (the noise-level factor). For continuous and discrete data values, the original data value is increased or decreased using a random value generated from a Gaussian distribution such that the new value is within the range of the values of that particular feature. The Gaussian distribution has zero mean; the standard deviation (second parameter) is estimated based on the data points.

To illustrate the concept of Algorithm 1, we inserted noises to the top ten features of the native model and trained a *noised model*. This noised model is also used in the rest of the paper for comparison purposes. Figure 5 shows how the feature ranks of the native model are changed due to noise insertion. The figure lists the top twenty important features of the native (left side of figure) and the noised (right side of figure) models. The feature `domain_occurrence` which is the most important feature in the native model is moved out of the top twenty in the noised model; and instead, `internal_link_ratio` feature (which has fourteenth rank in the native model) takes the top rank. Therefore, the evasion attack which targets the top twenty features of the native model is only able to hit less than half of them in the top twenty of the noised model. The evasion attack faces two different conditions. First, the sum of the feature importance of the attacked features is now much lower. However, (unavoidably) some other lesser important features have more importance with higher ranks. Second, the features which are not used for evasion have more feature importance in the noised model than the native model. The eleven features which are not targeted in the native attack have higher ranks in the noised model. We will show how these differences affect the performance against the evasion attack in Section IV.

Algorithm 1 Noise insertion to values of a target feature

```
Input:   $\mathbf{V} = \{v_i\}_{i=1}^{N}$;   $p$: noise-level factor
Output:  $\mathbf{W} = \{w_i\}_{i=1}^{N}$

 1: function InsertNoise($\mathbf{V}$, $p$)
 2:     $r_{\max} \leftarrow \max(\mathbf{V})$          ▷ upper bound of the values
 3:     $r_{\min} \leftarrow \min(\mathbf{V})$          ▷ lower bound of the values
 4:     if IsBinary($\mathbf{V}$) then
 5:         for $i \leftarrow 1$ to $N$ do
 6:             $d \leftarrow$ Bernoulli($p$)       ▷ Bernoulli distribution
 7:             if $d = 1$ then
 8:                 $w_i \leftarrow v_i \oplus 1$          ▷ flip when $d = 1$
 9:             end if
10:         end for
11:         return $\mathbf{W}$
12:     end if
        ▷ below steps, for both continuous and discrete numbers
13:     $g \leftarrow (r_{\max} - r_{\min}) \times p$
14:     for $v_i \in \mathbf{V}$ do
15:         $d \leftarrow$ Gaussian($0, g$)
16:         if IsInteger($\mathbf{V}$) then       ▷ for discrete numbers
17:             $d \leftarrow$ Round($d$)
18:         end if
19:         $w_i \leftarrow \min(r_{\max}, \max(r_{\min}, v_i - d))$
20:     end for
21:     return $\mathbf{W}$
22: end function
```

B. Building multiple models having diverse feature rankings

Towards building a robust detection system that is not adversely affected by evasion attacks, we build on our insight of randomizing the ranking of features. While the previous section described how we add noise to feature values, in this section, we decide on how to choose the features to be noised. Observe that, while inserting noise to feature values makes the model robust, that would also affect the classification accuracy (in terms of precision and recall). Therefore, robust model building has to consider the trade-off between accuracy under normal scenario and accuracy under evasion attack.

Considering the above, we next present Algorithm 2, that, in essence, selects a random number of features from each subset (or *bucket*) of features and inserts random noise into the corresponding values, to subsequently create multiple models with diverse feature ranking. The buckets form a partition of the entire feature set. If m denotes the number of features used to build a phishing detection model, let $k, k \leq m$, denote the number of features we want to select to define each bucket; therefore the size of each bucket is m/k (assuming m is a multiple of k). Therefore, if $\mathbf{F} = \{f_1, f_2, \ldots, f_m\}$ denotes the feature set used, the buckets b_i's are such that,

$$\cup_{i=1}^{(m/k)} b_i = \mathbf{F} \text{ and } \cap_{i=1}^{(m/k)} b_i = \emptyset.$$

Algorithm 2 takes as input the training dataset $\mathcal{D}^{\text{train}}$, the number of features m, the constant size k of all buckets, the number of features l_i to be noised in each bucket $b_i, 1 \leq i \leq m/k$, and the noise-level factor p_f for each feature f. The output is a model $\mathcal{M}$ that has $l_1 + l_2 + \ldots l_{m/k}$ features noised. Note that, each execution of Algorithm 2 gives a (possibly) different model with not only different number of features selected for noise insertion, but also trained on different feature values for those noised features. Thus we generate n models, $\mathcal{M}_1, \ldots, \mathcal{M}_n$ by running Algorithm 2 n times.

Algorithm 2 Model creation with random feature ranking

```
    Input: $\mathcal{D}^{\text{train}}; m; k; \{l_i\}_{i=1}^{(m/k)}; \{p_i\}_{i=1}^{m}$
    Output: $\mathcal{M}$

 1: for each bucket $b_j, j = [1, \dots, m/k]$ do
 2:     $\mathbf{F}^{\text{noise}} \leftarrow \mathbf{F}^{\text{noise}} \cup$ {Select $l_j$ features randomly from $b_j$}
 3: end for
 4: for $i \leftarrow 1$ to $m$ do
 5:     if $i \in \mathbf{F}^{\text{noise}}$ then
 6:         $\mathcal{D}_i^{\text{noise}} \leftarrow$ InsertNoise($\mathcal{D}_i^{\text{train}}$, $p_i$)
 7:     else
 8:         $\mathcal{D}_i^{\text{noise}} \leftarrow \mathcal{D}_i^{\text{train}}$
 9:     end if
10: end for
11: $\mathcal{M}$ = TrainModel($\mathcal{D}^{\text{noise}}$)
```

C. Robust phishing detection with multiple models

Once we generate multiple models with different feature ranking, we consider two approaches to combine them to build a robust phishing detection system. (i) We build n randomly generated models based on Algorithm 2, and then during operation (or testing) do the following: for each input (URL) provided for classification, pick one of n models randomly for classifying the given input, and take the output of that model as the classification result. While being simple, this approach does not allow the adversary to guess which model is being used for classification, thus weakening the evasion attack. (ii) In a second approach, we use a voting system that decides based on the output of each of the models; that majority prediction is the predicted class (phishing or benign) by the voting system. We refer to the first system as $\mathcal{R}$ and the second system as $\mathcal{V}$.

We note that building and operating multiple classifiers to have randomness against the exploratory attack is a known approach in machine learning; similar approaches are also used, say in large enterprises, wherein multiple detection solutions from different vendors are deployed for improved accuracy. However, to be effective, such approaches often require a large number of useful features as well as different datasets for training multiple classifiers. The novelty of our methodology is that it generates multiple randomized models based on a single feature vector and a given fixed training set, and these generated models are used to build a robust multi-classifier detection system.

IV. Evaluation

In this section, we evaluate the effectiveness of training with noises in enhancing robustness of the phishing URL classifiers against evasion attacks. We set multiple attack scenarios varying the evasion strategies based on the effort and knowledge of the adversary. To this end, we use AUPRC (area under precision-recall curves) as the metric for evaluation.

A. Experimental setup

Dataset:

We collected the list of benign and phishing URLs from public repositories. We obtained Alexa top websites [2] ranked in February 2019 and collected 110,090 HTML pages sampling from the top 300,000 domains by crawling from May 10 to August 5, 2019. The phishing URL list is collected from the daily feed of PhishTank database [38] between May 30 and July 10, 2019. For the PhishTank daily feed, we crawled the accessible phishing pages on the same day, resulting in a total of 32,159 phishing pages[3]. The full list of features used in this paper are given in Appendix.

We use 90% of dataset as a training set and the rest as a test set. Our dataset is imbalanced, with the ratio of benign to phishing being 1:3.4; we maintain the same ratio for our test sets. In practice, the phishing classification is a highly imbalanced class problem — we understand from security practitioners that 1:1000 is a possible ratio (a recent work estimates around 3:1000 [47]). We also highlight that, for imbalanced class problems, precision and recall (along with AUPRC) are better metrics than ROC curves. Indeed, it is well-known that ROC curves are possibly misleading metrics for imbalanced class problems [11]; while we have confirmed it for phishing detection as well, we omit the results illustrating the same.

Classification models:

(i) *native model:* The traditional ML model, referred to as the *native model*, is a phishing page detection classifier trained using Random forest with fifty-one URL and HTML features as mentioned in Section II-A.

(ii) *noised model:* We first evaluate the *noised model* which is defined in Section III-A; $\mathbf{F}^{\text{noise}}$ is the list of top ten important features of the native model. The noised model is identical to the model created by Algorithm 2 with $k = 10, \{l_i\}_{i=1}^{5} = [10, 0, 0, 0, 0]$; i.e., noises are inserted only to the top ten features of the native model.

(iii) *randomly generated models:* We create 11 models by running Algorithm 1 as many times. Based on the intuition that inserting noise to more features in a bucket changes the ranks of the corresponding features, while drop in the feature importance of high-ranked features will degrade detection accuracy, we set diverse values to l_i's for creation of each model. (The last of the 51 features has always had close to zero importance; hence we ignore it.) Experimenting with different values of l_i's, we found that inserting noise into the last two buckets do not make meaningful changes. We keep $l_3 = 5$, and then choose values for the first two buckets such that the minimum value of l_1 is 5, and the values of l_2 are between 0 to 10 so as to have the sum of l_1 and l_2 between 15 and 19. Thus the 11 combinations for l_i's we consider are: five sets $[5, 5, 5, 0, 0]$, $[6, 4, 5, 0, 0]$...$[9, 1, 5, 0, 0]$, another five sets $[5, 9, 5, 0, 0]$, $[6, 8, 5, 0, 0]$...$[9, 5, 5, 0, 0]$, and the noised model $[10, 0, 0, 0, 0]$.

Evasion method and strategies:

An adversary can afford to learn the feature set from existing studies, literature, by training a ML model, or, in the worst case by reverse-engineering the detection system [27]. To build an ML model, an adversary can use the publicly available datasets (Alexa [2] for benign, and PhishTank [38] or

[3]The datasets used in this paper for model building are available in the public repository https://github.com/JehLeeKR/phishing-madweb/.

OpenPhish [36] for phishing URLs, for example) for training and subsequently learning the important features.

The evasion attacks we consider for experimentation here are basically achieved by copying the feature values from the benign dataset. In the case of phishing, this is also what an attack would want to achieve — to mimic a (target) benign page as much as possible. And benign pages are (obviously) available in public. In our experiments, the target benign pages that are mimicked are picked randomly, and the attack is constructed at the feature level — that is, features of a phishing URL is mimicked using the corresponding values from a benign feature vector. For maintaining the correlation between the feature values, all mimicked feature values of a phishing page come from the same benign page. However, note that, copying values of all benign features goes against the attack itself; for example, the phishing URL will not be the same as the targeted benign URL.

We define below, the different evasion strategies we assume an adversary can use against the ML-based phishing detection systems.

- **[S1] Random feature attack:** If an adversary cannot make an informed guess on the important features of the phishing classifier, the simplest strategy is to evade using an arbitrary set of features. In this random feature attack, we allow random selection of 20 features from the full list of 51 features.
- **[S2] Evasion assuming knowledge of model-generation algorithm:** We assume adversary has knowledge of Algorithm 2 we defined for model generation. However, the adversary has no knowledge of the detection system in operation; in particular, how many models are used in operation and what kind of detection approach is used (randomly picked model or voting) is unknown to the adversary. Therefore, the adversary generates one model using Algorithm 2, and uses the top 20 features of this model for generating evasive samples. For a detection system that uses the same model among one of the randomly generated models (which is the case we consider), the attacker is guaranteed to 'hit' the right model at least once.
- **[S3] Evasion assuming knowledge of the algorithm and partial knowledge of operation:** We assume a stronger adversary who, not only has knowledge of Algorithm 2, but also has partial knowledge of detection system during operation. The attacker here knows the number of randomly generated models used for building the detection system; but yet does not know the defense strategy (random model or voting). The adversary, therefore, trains 11 models (as is used in our detection system in the experiments below), and subsequently finds out the 20 most frequently appearing features among the top 20 features of all the models. These most frequent 20 features are then used to generate evasive phishing samples. This is based on the intuition that, if a feature appears more frequently at high ranks in the multiple models, it is more likely to evade a running model.

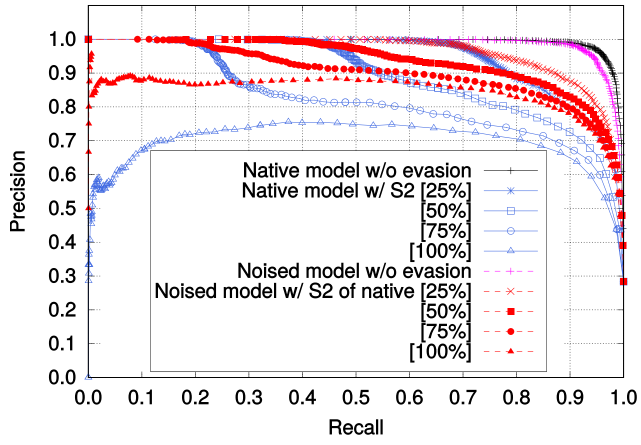


Fig. 6: Robustness of noised model against attack evading the top 20 features (this is the same scenario used to illustrate the attack on the native model in Figure 2)

B. *Results: single noised model*

We analyze the single noised model, where noises are inserted into the top 10 features. In Figure 6, we compare this model with the native model, under normal scenario as well as under evasion attack where the top 20 features are evaded. The legend [x%] means, x% of the phishing test set was modified for evasion. We observe that the noised model is much more robust than the native model, against the evasion attack. Yet, the AUPRC of the noised model is very close to the native model when there is no evasion.

However, for a fair comparison, the noised model should also be tested against an attack that evades on the top 20 features of the noised model (recall that, the feature ranking changes when noise is inserted; and the evasion features used above was the top 20 features of native model). Our experiments showed that the noised model is weak against attack **[S2]** that used top features from the noised model. The AUPRC for 50% test evasion reduced from a high 0.987 (under no evasion) to 0.791 (under evasion). This is not surprising — given that the noised model is just another model with a different ranking of features, it will also be weak against an evasion attack targeting its own top features. For achieving such a weak performance, the adversary has to build a noised model on his own and then target the top features. Therefore, building yet another model that changes the feature ranks does not help against evasion attacks.

From now on, we show results for the case where 50% of the phishing test set is modified for evasion.

C. *Results: detection system with multiple randomized models*

The key property we try to achieve with the multi-model detection system is to have diverse feature ranking across the different models generated. To validate, we generated 11 models using Algorithm 2. We now analyze the feature ranks of the top 10 features of native model, in each of these 11 models. Figure 7 plots the feature rankings. Observe that the top 10 features of the native model (marked on the Y-axis) are distributed across a wide range of ranks for each model, confirming to our goal of randomizing feature-rankings across

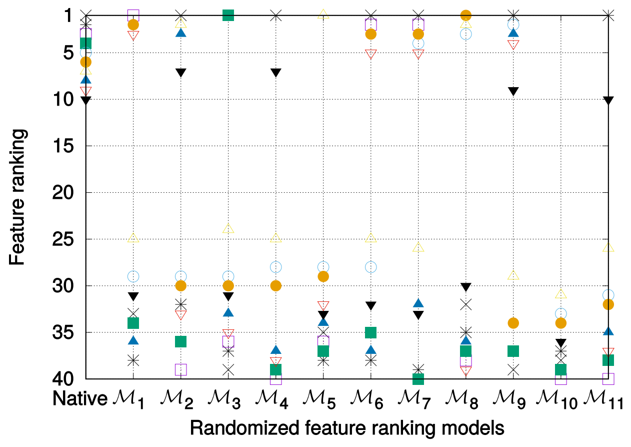


Fig. 7: Rankings of top 10 features of the native model in randomly generated models $\mathcal{M}_1 ... \mathcal{M}_{11}$, following Algorithm 2.

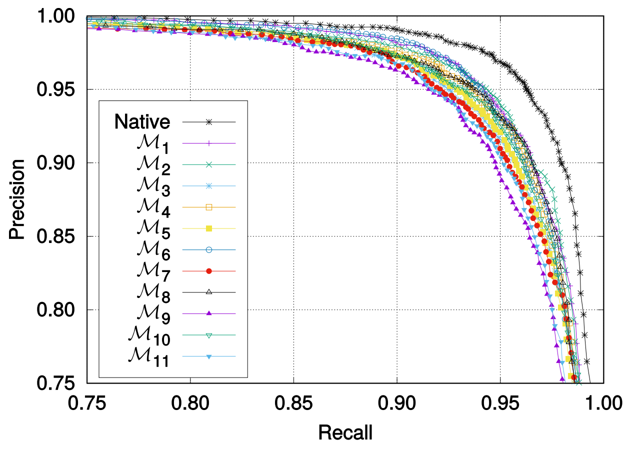


Fig. 8: Performance comparison of randomly generated models with the native model. Note, for better clarity, the X and Y ranges are reduced to the higher end of the spectrum.

models. Recall that, for these models $l_1 \geq 5$; hence at least five of the top 10 features of the native model are selected for noise insertion in each of these randomly generated models.

Before testing the robustness of the multi-model detection systems, we first evaluate the accuracy of each of the randomly generated models; this is necessary as the feature rankings in these models are different from the native model. More importantly, random number of features are noised in each of these models. As the precision-recall curves in Figure 8 confirm, the performance degradation by noise insertion is small. The recall for 0.95 precision for the native model is 0.96, whereas that for the randomly generated models are in the range [0.92, 0.94].

As discussed in Section III-C, we build two phishing detection systems based on the multiple models generated. The first one, $\mathcal{R}$, chooses a model randomly each time a URL is processed, whereas the second one, $\mathcal{V}$, uses voting to decide on the URL. Evidently, $\mathcal{R}$ requires execution of only one model to make a decision, whereas $\mathcal{V}$ requires execution of all models for an input URL. Figure 9 plots the performance of $\mathcal{R}$ under the three evasion strategies. As there are 11 models, we plot the

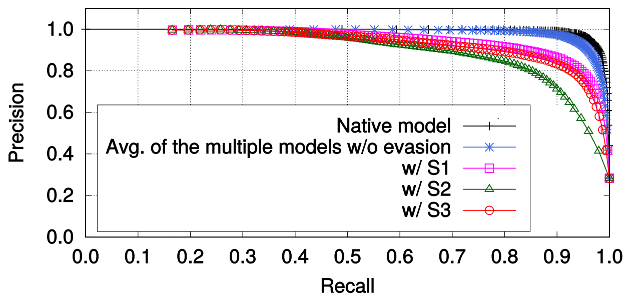


Fig. 9: Detection performance of the multiple models in average, against the three evasion attacks considering multi-model system ($\mathcal{R}$)

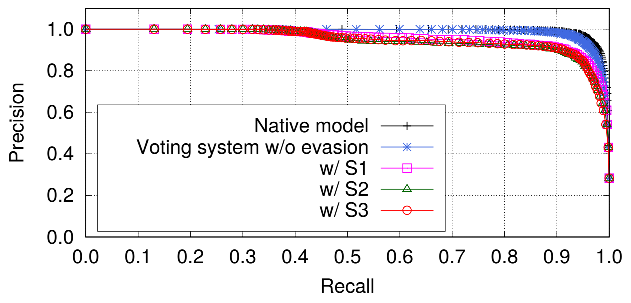


Fig. 10: Detection performance of a voting system $\mathcal{V}$ against the three evasion attacks

average of the 11 models in this figure. Under both **[S1]** and **[S3]**, for a precision of ~ 0.9, a recall of 0.8 can be achieved. For **[S2]**, the recall at 0.9 precision drops to 0.7, which is still significantly better than what the native model achieves (around 0.55 recall at 0.9 precision). Interestingly, while **[S3]** requires more knowledge of the system in operation than **[S2]**, the latter is a more effective attack than the former.

Figure 10 plots the performance of the voting system $\mathcal{V}$. Clearly, it performs better than $\mathcal{R}$ that selects models randomly in operation. Under all attacks, $\mathcal{V}$ achieves a recall of 0.9 for ~ 0.9 precision. We also provide the AUPRC values of the native model and detection systems $\mathcal{R}$ and $\mathcal{V}$ in Table II. When there is no evasion attack, all three models have similar performance. The random attack **[S1]** does not degrade the accuracy of any of the models. Whereas under **[S2]**, the voting-based multi-model detection system outperforms both the native model as well as the detection system $\mathcal{R}$ that randomly picks one of the models during operation.

Finally, we experiment in a scenario where *the attacker chooses different models against a deployed detection system*. The evasion strategy is a generalization of **[S2]** — attacker generates 11 models (as we do for generating the randomized models) following Algorithm 2, thus creating models $\mathcal{M}_1, \mathcal{M}_2, \ldots \mathcal{M}_{11}$. In one experiment, the detection system tested is a static model (one of these 11 models is selected randomly for this purpose). In the second experiment, the voting-based multi-model detection system $\mathcal{V}$ is used; that is, we test against $\mathcal{V}$ which uses majority voting among the same 11 models $\mathcal{M}_1, \mathcal{M}_2, \ldots \mathcal{M}_{11}$ to classify a URL. Figure 11 plots the results against the static model. The worst-case performance is what happens if the attacker is "lucky".

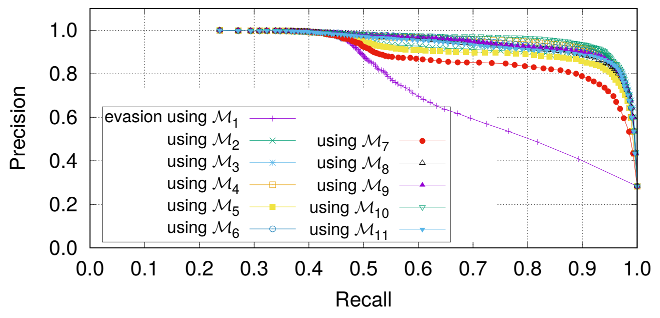


Fig. 11: Detection performance of a detection system with a static model when attacked using a generalized **[S2]** evasion strategy that uses features from random models to evade

TABLE II: AUPRC of multi-model systems against sophisticated evasion attacks

Evasion type	Native model	$\mathcal{R}$	$\mathcal{V}$
w/o evasion	0.991	0.983	0.987
S1	0.972	0.945	0.966
S2	0.892	0.900	0.947
S3	N/A	0.932	0.950

For precision of 0.8, the corresponding recall achieved is approximately 0.5. This worst-case performance is noticeable in the static model system with any of the 11 models as well as with the native model. Whereas, as seen in Figure 12, for the same attack against the voting-based detection system $\mathcal{V}$, the recall at 0.8 precision in the worst-case is a high 0.84. This experiment demonstrates that, the voting-based multi-model detection system is not only better on average, but is much better in the worst-case than a statically chosen model generated via Algorithm 2.

V. Discussions and Future Work

Multi-model evasion with base model knowledge: We now assume the strongest adversary. Though Algorithm 2 generates different models at each execution, they all are essentially selecting a set of features from a given static set of 51 features for noise insertion. This static set of features implicitly defines a *base* model built from the feature set without any noise, and all the generated models are related to this base model as the original feature vector is fixed. Even if an adversary has knowledge of Algorithm 2, it will be extremely difficult to figure out this base model (this would require breaching the machine where the model is trained and successfully performing runtime leak of the program), along with the fact that there are exactly 11 models used in operation. Yet, we assume such an adversary and evasion strategy here, to further test our proposed detection system. The results are plotted in Figure 13. The comparison of our proposed multi-model detection systems is done with the native model. Note, for the native model, the base model is the same as the native model. We observe that the voting-based detection system performs the best. For precision of 0.8, the recall achieved is 0.9. Whereas, for the native model, 0.8 precision gives a recall of less than 0.8.

Black-box attack to the multi-classifier system: Black-box attack is a part of our threat model to learn the important

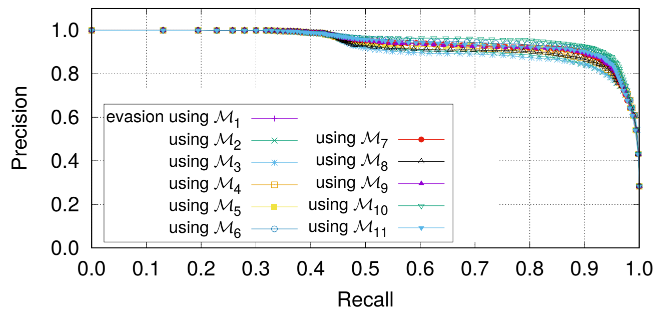


Fig. 12: Detection performance of the voting-based system ($\mathcal{V}$) when attacked using a generalized **[S2]** evasion strategy that uses features from random models to evade

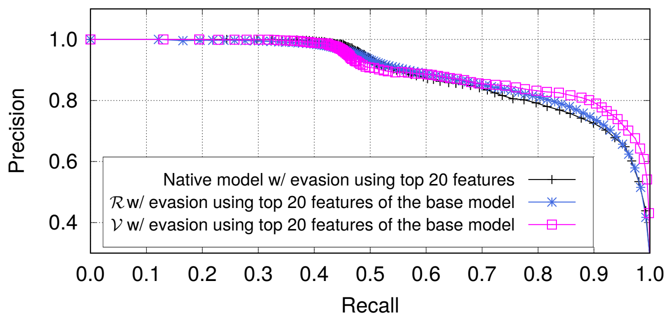


Fig. 13: Detection performance of a multi-model system ($\mathcal{R}$) and a voting-based system ($\mathcal{V}$) compared to the native model when attacked using top 20 feature of the base model used to create the multiple models

features, yet, the black-box attack also allows an adversary to have his own substitute model [50], [37]. As it has been reported that an ensemble of classifiers does not guarantee robustness against adversarial samples [20], [3], the attacks with adversarial examples based on the system output (instead of the important features learned) should be considered as a threat to our proposal. This sophisticated attack would be considered for future work.

Evasion and noise insertion in Support Vector Machine: Our proposed robust phishing detection system was based on Random forest classifier, and so were the experiments until now. Another widely used ML model is Support Vector Machine (SVM); and there have been studies on multi-classifier approaches (e.g., SVM ensemble) for enhancing the robustness and accuracy of SVM-based solutions [24]. We now conduct a preliminary study to analyze the performance of SVM classifier for phishing detection, including under the evasion attacks. In addition, we also analyze how robust is a *noised* SVM model against evasion attacks.

Specifically, we built an SVM model with the radial basis function kernel and tested evasion attacks on 50% of the phishing test set, wherein evasion was based on (i) random 20 features and (ii) top 20 features of the native model. We used Algorithm 1 on the top 10 features of the native SVM model to create a noised model. The results, depicted in Figure 14, give interesting insights compared to the results from Random forest models. One, the performance of the native SVM model

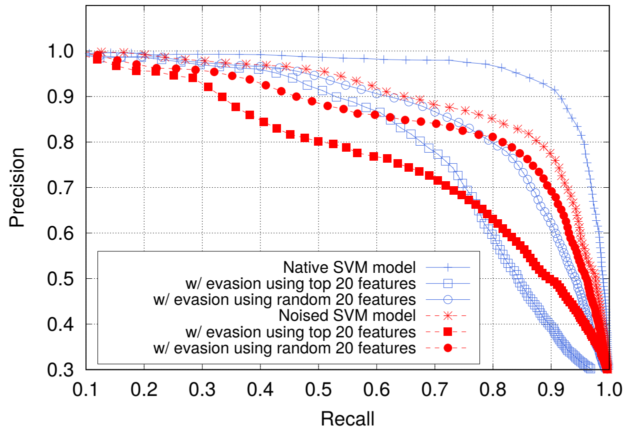


Fig. 14: Effect of evasion attacks and noise insertion on an SVM classifier; evasion on random 20 features and evasion on top 20 features from Random forest model to a native SVM and a noised SVM model

degrades significantly under evasion attacks. The degradation is more pronounced when the evasive features are selected from the top 20 features (based on Random forest classifier) than when randomly selected. This means, knowledge of the top features, even though it was based on Random forest classifier, is helpful for an attacker to evade detection by the classifier. Two, unlike in previous experiments (using Random forests), the noised model experiences non-negligible degradation in PRC (precision-recall curve) when noise is injected into the native model.

And similarly, under evasion attack, the noised model is not more robust than the native model. Thus the current noise insertion approach is not directly applicable to SVM. However, changing feature ranking is still a valid approach for improving the robustness of SVM classifiers [10], [15].

VI. Related Work

Adversarial examples and attacks targeting ML-classifiers are not only an issue in the security domain but in others as well [42], [9]. Meanwhile, security applications like phishing and malware detection systems are relatively less threatened by the causative attacks which target the training process and training data [25] due to the special attention and inspection on their training data; instead, their services in public are open and hence vulnerable to exploratory attacks [4], [23], [27]. There have been two streams of studies explored to combat exploratory attacks (which rely on the knowledge of an adversary); one is via robust modeling, while the other is by preventing learning capability of the adversaries.

Kolcz *et al.* [25] analyze and report the cost and performance of the robust modeling methods, in particular, by feature re-weighting. Of the few approaches for building a robust model by regularizing biased feature weights, the noise insertion approach [52], [33] can be considered suitable in domains that suffer from limited training set and risks not only accidental failure of a model but also deliberate evasion attacks (e.g., phishing page detection classifier).

On the other hand, the multi-classifier systems that are widely used for better accuracy and fault-tolerance [21], [41] are also considered as a countermeasure to prevent adversarial learning; and that is primarily based on the diversity and randomness achieved via the use of multiple models. Biggio *et. al.* studied randomisation strategies based on multi-classifier systems to prevent adversarial learning for evading a classifier, and also confirmed that a multi-classifier system can be more robust than a single classifier system in their series of studies [6], [7].

A recent study by Tong *et al.* [48] introduces the idea of conserved features that are not modifiable without compromising the intended malicious functionality. They propose an algorithm to automatically extract the conserved features from the feature set and show effectiveness against evasion attacks on PDF malware detection. While this is a promising approach to limit the adversary in launching an evasive attack, such features are limited in the case of phishing; for example, much of the HTML features of a phishing page can be very similar to that of a benign web page, without compromising attacker's goal.

VII. Conclusion

Application of machine learning for phishing detection has been a promising direction in protecting end-users. However, as demonstrated by recent works as well as in this paper, ML-based phishing classifiers are also prone to adversarial attacks. Based on the insight that an adversary, with sufficient knowledge of machine learning algorithms and access to typically used training datasets, can learn important features for phishing detection, we explored an approach to randomize feature ranking to build a robust phishing detection system. Our proposed methodology makes it possible to create a number of diverse models from the same training dataset as well as the same feature set, and subsequently build a meta-classification system. Evaluations on real dataset showed that the voting-based multi-model phishing detection system not only has similar performance as the native model when there is no evasion attack, but is also more robust than the native model under evasion attack.

Acknowledgement

This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd.

References

[1] (2019) Phishing trends and intelligence report. [Online]. Available: https://info.phishlabs.com/hubfs/2019PTIReport/2019PhishingTrendsandIntelligenceReport.pdf

[2] Alexa Internet, Inc, "Alexa top sites," https://www.alexa.com/topsites, 2019.

[3] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018. [Online]. Available: https://arxiv.org/abs/1802.00420

[4] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proc. ACM CCS*, 2006, pp. 16–25.

[5] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, "Bagging classifiers for fighting poisoning attacks in adversarial classification tasks," in *International workshop on multiple classifier systems*. Springer, 2011, pp. 350–359.

[6] B. Biggio, G. Fumera, and F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation," in *Proc. IAPR Joint Conf. S+SSPR*. Springer, 2008, pp. 500–509.

[7] ——, "Multiple classifier systems for robust classifier design in adversarial environments," *Intl. Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.

[8] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains," *ACM Trans. Inf. Syst. Secur.*, vol. 16, no. 4, Apr. 2014.

[9] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," in *Proc. IEEE SPW*, 2018, pp. 1–7.

[10] Y.-W. Chen and C.-J. Lin, "Combining SVMs with various feature selection strategies," in *Feature extraction*, 2006, pp. 315–324.

[11] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proc. ACM ICML*, 2006, pp. 233–240.

[12] O. Dekel, O. Shamir, and L. Xiao, "Learning to classify with missing and corrupted features," *Machine learning*, vol. 81, no. 2, pp. 149–178, 2010.

[13] B. A. Desai, D. M. Divakaran, I. Nevat, G. W. Peter, and M. Gurusamy, "A feature-ranking framework for IoT device classification," in *Proc. COMSNETS*. IEEE, 2019, pp. 64–71.

[14] A. Y. Fu, L. Wenyin, and X. Deng, "Detecting phishing web pages with visual similarity assessment based on earth mover's distance (EMD)," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 4, pp. 301–311, 2006.

[15] A. Globerson and S. Roweis, "Nightmare at test time: robust learning by feature deletion," in *Proc. ACM ICML*, 2006, pp. 353–360.

[16] Google Developers, "Google Safe Browsing." [Online]. Available: https://developers.google.com/safe-browsing/

[17] X. Guang, H. Jason, P. R. Carolyn, and C. Lorrie, "CANTINA+: A feature-rich machine learning framework for detecting phishing web sites," in *Proc. ACM TISSEC*, 2011, pp. 1–28.

[18] I. Hafeez, A. Y. Ding, L. Suomalainen, A. Kirichenko, and S. Tarkoma, "Securebox: Toward safer and smarter IoT networks," in *Proc. ACM CoNEXT*, 2016, pp. 55–60.

[19] X. Han, N. Kheir, and D. Balzarotti, "Phisheye: Live monitoring of sandboxed phishing kits," in *Proc. ACM CCS*, 2016, pp. 1402–1413.

[20] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *Proc. WOOT*, 2017.

[21] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 1, pp. 66–75, 1994.

[22] S. Hossein, B. Bruhadeshwar, and R. Indrakshi, "Kn0W Thy Doma1N Name: Unbiased Phishing Detection Using Domain Name Based Features," in *Access Control Models and Technologies*, 2018.

[23] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proc. ACM AISEC*, 2011, pp. 43–58.

[24] H.-C. Kim, S. Pang, H.-M. Je, D. Kim, and S. Y. Bang, "Constructing support vector machine ensemble," *Pattern recognition*, vol. 36, no. 12, pp. 2757–2767, 2003.

[25] A. Kołcz and C. H. Teo, "Feature weighting for improved classifier robustness," in *Proc. CEAS*. ACM, 2009.

[26] H. Le, Q. Pham, D. Sahoo, and S. C. Hoi, "URLNet: learning a URL representation with deep learning for malicious URL detection," *arXiv preprint arXiv:1802.03162*, 2018. [Online]. Available: https://arxiv.org/abs/1802.03162

[27] B. Liang, M. Su, W. You, W. Shi, and G. Yang, "Cracking classifiers for evasion: a case study on the Google's phishing pages filter," in *Proc. WWW*, 2016, pp. 345–356.

[28] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious URLs," in *Proc. ACM SIGKDD*, 2009, pp. 1245–1254.

[29] McAfee, "McAfee WebAdvisor." [Online]. Available: https://www.mcafee.com/consumer/en-sg/store/m0/catalog/mwad_528/mcafee-web-advisor.html

[30] D. K. McGrath and M. Gupta, "Behind Phishing: An Examination of Phisher Modi Operandi." vol. 8, 2008.

[31] E. Medvet, E. Kirda, and C. Kruegel, "Visual-similarity-based phishing detection," in *Proc. SecureComm*. ACM, 2008.

[32] Mimecast, "Advancing Your Anti-Phishing Program," 2018. [Online]. Available: https://www.gartner.com/imagesrv/media-products/pdf/mimecast/Mimecast-1-4QT9Y3H.pdf

[33] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," *arXiv preprint arXiv:1511.06807*, 2015. [Online]. Available: https://arxiv.org/pdf/1511.06807.pdf

[34] I. Nevat, D. M. Divakaran, S. G. Nagarajan, P. Zhang, L. Su, L. L. Ko, and V. L. L. Thing, "Anomaly Detection and Attribution in Networks With Temporally Correlated Traffic," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 131–144, Feb 2018.

[35] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection," in *IEEE Conf. on Communications and Network Security (CNS)*, June 2019, pp. 91–99.

[36] OpenPhish, "OpenPhish," https://openphish.com/, 2019.

[37] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM CCS*, 2017, pp. 506–519.

[38] PhishTank®, "PhishTank," https://phishtank.com/, 2019.

[39] V. Rakesh and D. Keith, "On the character of phishing urls: Accurate and robust statistical learning classiers," in *Proc. ACM CODASPY*, 2015, pp. 111–122.

[40] B. D. Rouani, M. Samragh, T. Javidi, and F. Koushanfar, "Safe machine learning and defeating adversarial attacks," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 31–38, 2019.

[41] D. Ruta and B. Gabrys, "Classifier selection for majority voting," *Information fusion*, vol. 6, no. 1, pp. 63–81, 2005.

[42] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting classifiers against adversarial attacks using generative models," *arXiv preprint arXiv:1805.06605*, 2018. [Online]. Available: https://arxiv.org/abs/1805.06605

[43] M. Samuel, F. Jrme, S. Radu, and E. Thomas, "PhishStorm: Detecting phishing with streaming analytics," vol. 11, no. 4, pp. 458–471, 2014.

[44] M. Samuel, S. Kalle, S. Nidhi, and A. N, "Know Your Phish: Novel Techniques for Detecting Phishing Sites and Their Targets," in *Proc. IEEE ICDCS*, 2016.

[45] H. Shirazi, B. Bezawada, I. Ray, and C. Anderson, "Adversarial sampling attacks against phishing detection," in *Data and Applications Security and Privacy XXXIII*, S. N. Foley, Ed., 2019, pp. 83–101.

[46] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time URL spam filtering service," in *Proc. IEEE S&P*, 2011, pp. 447–462.

[47] K. Tian, S. T. K. Jan, H. Hu, D. Yao, and G. Wang, "Needle in a haystack: Tracking down elite phishing domains in the wild," in *Proc. IMC*. ACM, 2018, pp. 429–442.

[48] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, and Y. Vorobeychik, "Improving robustness of ML classifiers against realizable evasion attacks using conserved features," in *Proc. USENIX Security*, 2019, pp. 285–302.

[49] V. Total, "VirusTotal-free online virus, malware and URL scanner," 2012. [Online]. Available: https://www.virustotal.com/en

[50] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proc. USENIX Security*, 2016, pp. 601–618.

[51] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages," in *Proc. NDSS*, 2010.

[52] Z. You, J. Ye, K. Li, Z. Xu, and P. Wang, "Adversarial noise layer: Regularize neural network by adding noise," in *IEEE ICIP*, 2019, pp. 909–913.

[53] L. Yukun, Y. Zhenguo, C. Xu, Y. Huaping, and L. Wenyin, "A stacking model using URL and HTML features for phishing webpage detection," in *Future Generation Computer Systems*. Elsevier, 2019, pp. 27–39.

APPENDIX

FEATURE LIST

TABLE III: List of features used in our work here

Index	Source	Feature	Explanation (refer to the cited works for further details)
1		is_domain_ip	If hostname is an IP address
2		num_subdomain_level	Number of subdomain levels
3		has_embedded_domain	If the path part of a URL contains dot separated domain/hostname patterns [17]
4		num_url_tokens	Number of tokens in URL (non-alphanumeric characters)
5		has_sensitive_word	If the URL contains any of [‘secure’, ‘account’, ‘webscr’, ‘login’, ‘ebayisapi’, ‘signin’, ‘banking’, ‘confirm’] [17]
6		is_obfuscated_url	If IP addresses is represented in hex or octet format, or embedding path-traversal operations in a URL string. [46]
7		len_domain	Length of domain name
8	URL	len_path	Length of path
9		len_url	Length of URL
10		num_url_dots	Number of dots in the URL
11		has_suspicious_char_url	If the URL contains ‘@’ or ‘∼’, or the domain name contains ‘-’ [17]
12		has_tld_out_of_position	If TLD is out of position [17]
13		len_fqdn	Length of the FQDN
14		len_mld	Length of the main level domains (MLD) [43]
15		num_mld_terms	Number of terms in the main level domains [43]
16		has_bad_forms	If mutual satisfaction of following four conditions [17]: 1. existence of <form>, 2. <input>tag in the <form>, 3. keywords related to password/credit card number or no text at all but images only within the scope of HTML form 4. a non-https scheme in the URL in the action field or in the webpage URL when the action field is empty.
17		num_terms_in_text	Number of space delimited terms in text
18		num_terms_in_title	Number of space delimited terms in <title>
19		num_input_fields	Number of <input>tags tags in the page
20		num_images	Number of <img>tags in the page
21		num_iframes	Number of <iframe>tags in the page
22		num_links	Number of <link>tags in the page
23		ratio_internal_link	Ratio of links under the same domain with the URL
24		num_empty_link	Number of following tag patterns [53]: <a href = “”></a>, or <a href = “#” ></a>, or [‘#javascript::void(0)’, ‘#content’, ‘#skip’, ‘javascript:;’ , ‘javascript::void(0);’, ‘javascript::void(0)’]
25		has_login_form	If <form>tag <input>sub-tag contain password or ‘pass or ‘login or ‘signin’ [53], [51]
26		len_html_style	Length of HTML content in <style>
27	HTML	len_html_script	Length of HTML content in <script>
28		len_html_comment	Length of HTML content in <!−− >
29		len_html_form	Length of HTML content in <form>
30		has_alarm_window	If <script>tag contains ’alert’ or ’windows.open’ [53]
31		has_hidden_content	If contains any of the below [53]: <div>: <div style = “visibility: hidden” >, or <div style=“display: none>. <button>: <button disabled = “disabled” >. <input>: <input type =“hidden” >, <input disabled =“disabled” >, <input value =“hello” >
32		has_brand_in_title	If *brand* [44] is in <title>
33		ratio_internal_resource	Ratio of use of identical domain name in resources, linked URLs and embedded page URLs
34		num_brand_occurrence	Times the *brand* appearing in HTML [44]
35		len_html_body	Length of HTML code including tags and plain text
36		len_html_plain_text	Length of HTML plain text
37		max_dom_depth	Maximum depth of DOM trees
38		dom_node_count	Number of the end nodes (leaves) of DOM trees
39		dom_node_type	Number of the unique end node types
40		dom_node_mean	Mean of the end nodes’ depth
41		dom_node_std	Standard deviation of the end nodes’ depth
42		ratio_link_same_page	Ratio of the hyperlinks or the resources’ links pointing to the same page
43		ratio_link_same_folder	Ratio of the hyperlinks or the resources’ links pointing to the same folder
44		num_unique_subdomain	Number of unique sub-domains in HTML
45		num_unique_file_type	Number of unique file types in HTML
46		num_sub_directory	Number of sub-directory
47		num_unique_directory_path	Number of unique directory path
48		directory_depth_mean	Mean of the depth of directory paths
49		directory_depth_std	Standard deviation of the depth of directory paths
50		min_dist_top_words	Minimum distance between top 5 words from HTML plain text and the words in each level of the domain name
51		ratio_script_in_tags	Ratio of script tags out of all the tags