Phishing Detection Leveraging Machine Learning and Deep Learning: A Review

Dinil Mon Divakaran | Trustwave
Adam Oest | PayPal

Phishing attacks trick victims into disclosing sensitive information. To counter them, we explore machine learning and deep learning models leveraging large-scale data. We discuss models built on different kinds of data and present multiple deployment options to detect phishing attacks.

Phishing attacks attempt to deceive users with the goal of stealing sensitive information, such as user credentials (usernames and passwords), bank account details, and even entire identities. To achieve this, as illustrated in Figure 1, an attacker deploys a malicious website that resembles the legitimate website of a well-known target brand. Figure 2 shows a real phishing page targeting Facebook. Subsequently, the attacker sends a link to the phishing website to potential victims and leverages social engineering techniques to lure those victims into disclosing confidential information. Figure 1 illustrates the general process, which can be quite complicated in practice. For example, a recent phishing campaign used a compromised email account to send victim users a malicious attachment containing obfuscated code. This code, in turn, retrieved dynamic scripts from a hosting server that ultimately generated HTML to render a phishing page targeting Microsoft Office 365 users (for a detailed description, see Pacag[1]).

Indeed, phishing attacks have evolved to increasingly use shortened uniform resource locators (URLs), redirection links, the HTTPS protocol, server- and client-side cloaking techniques (displaying benign pages to antiphishing crawlers), and so on to evade detection.[2, 3] To add to these, today there exist software tool kits, or phishing kits, in dark markets, that automate the majority of the aforementioned process. Thus, it is no surprise that phishing continues to be one of the top cyberattacks today. Furthermore, the COVID-19 pandemic and prevalence of remote work have given rise to new social engineering and victimization opportunities for attackers.[4]

Antiphishing Blacklists

A well-known technique for protecting users from accessing phishing webpages is to use an antiphishing blacklist. Webpage URLs accessed from a modern web browser (e.g., Google Chrome, Mozilla Firefox, and Microsoft Edge) are automatically checked against a list of known phishing URLs, and access to matched URLs is warned/blocked. URLs that ultimately end up on antiphishing blacklists are collected, crawled, and analyzed by antiphishing entities (e.g., Google Safe Browsing, the Anti-Phishing Working Group, and PhishTank), each of which leverages a variety of

Digital Object Identifier 10.1109/MSEC.2022.3175225
Date of current version: 14 June 2022

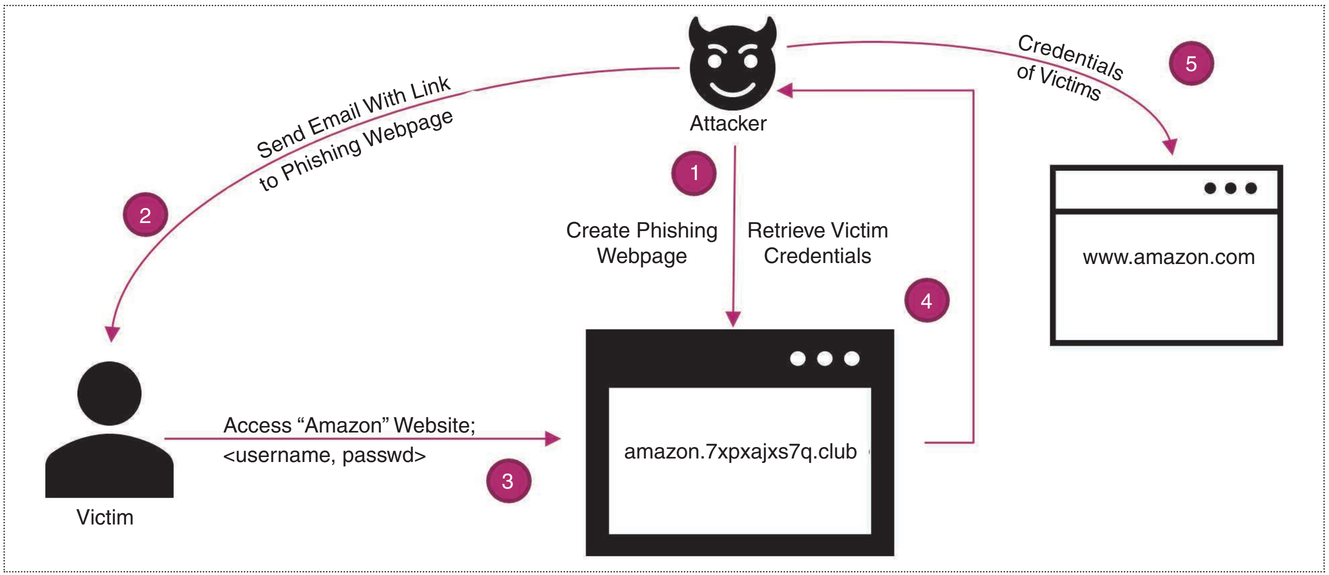
Attacker
1
Create Phishing Webpage
Retrieve Victim Credentials
2
Send Email With Link to Phishing Webpage
Victim
Access "Amazon" Website; <username, passwd>
3
amazon.7xpxajxs7q.club
4
Credentials of Victims
5
www.amazon.com

Figure 1. The phishing process. 1) An attacker deploys a phishing webpage mimicking a well-known legitimate target, e.g., Amazon. 2) The attacker sends the corresponding link to numerous potential victims via emails, social networks, and so on. 3) A user who gets deceived accesses the phishing webpage and provides sensitive information. 4) The user's credentials are retrieved by the attacker 5) to access the targeted website to commit fraud.

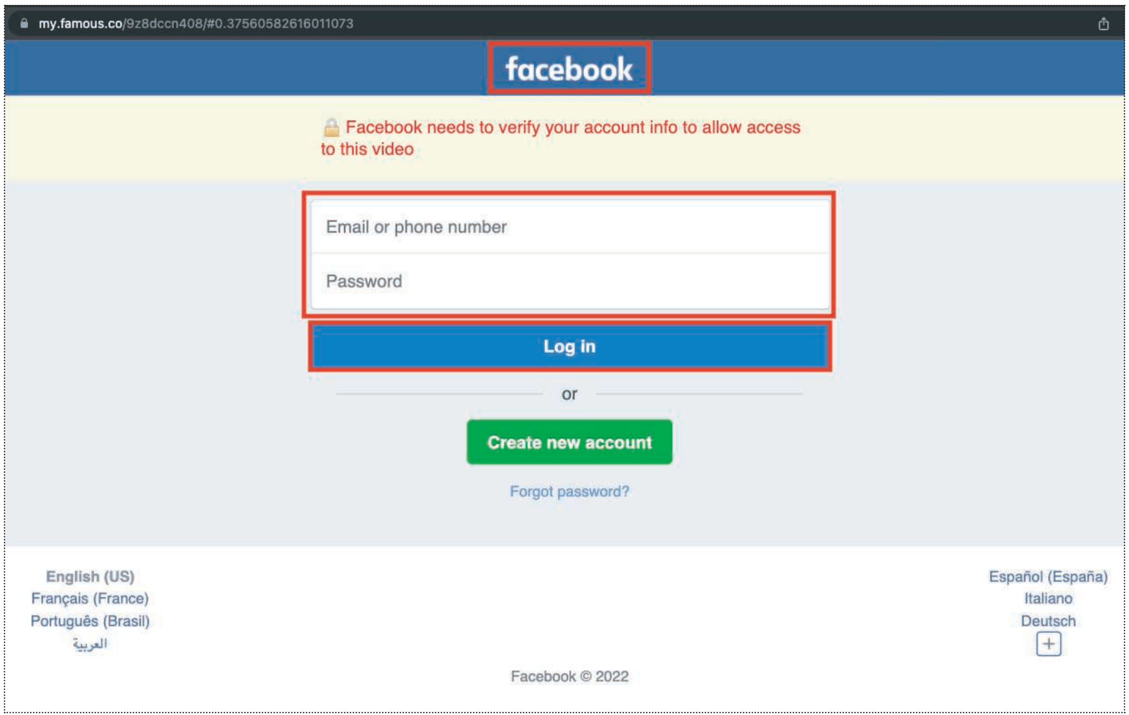

Figure 2. A phishing page targeting Facebook (screenshot taken in March 2022). Red boxes highlight important phishing characteristics—the logo, input form, and login button—designed to deceive a user.

threat intelligence sources and cross-organizational partnerships.

Blacklists are efficient and scale well with the number of phishing webpages in the wild. A URL lookup can easily be done in real time, employing a user's computer resources, and such lookups also preserve user privacy. However, antiphishing blacklists are not themselves capable of detecting new and unseen phishing pages:

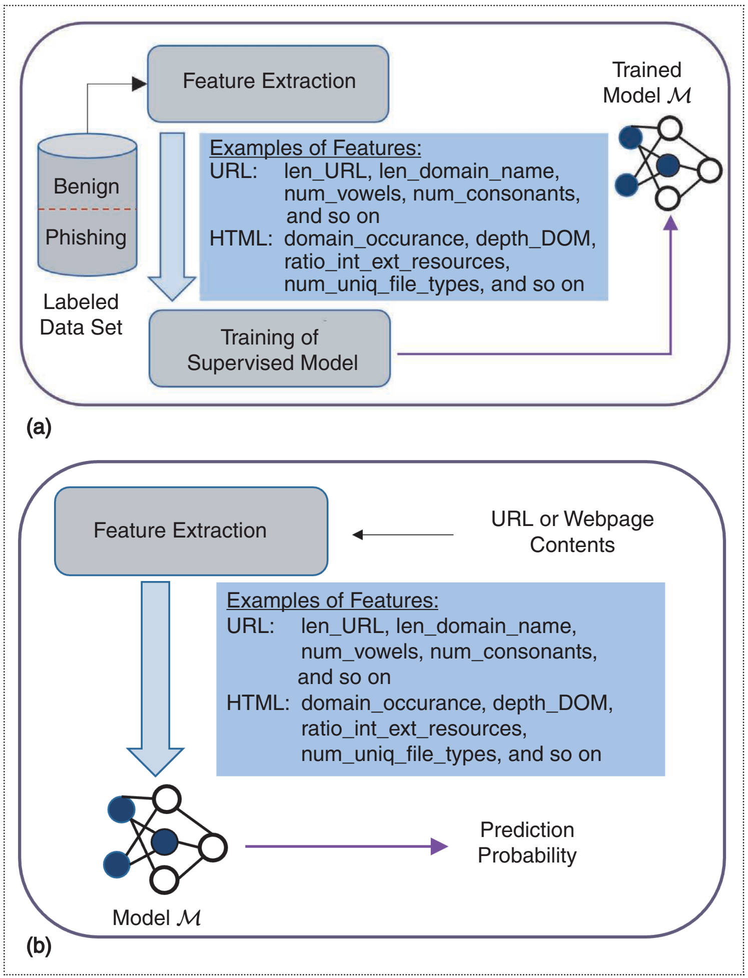
Feature Extraction
Trained Model $\mathcal{M}$
Benign
Phishing
Labeled Data Set
Examples of Features:
URL: len_URL, len_domain_name, num_vowels, num_consonants, and so on
HTML: domain_occurance, depth_DOM, ratio_int_ext_resources, num_uniq_file_types, and so on
Training of Supervised Model
(a)
Feature Extraction
URL or Webpage Contents
Examples of Features:
URL: len_URL, len_domain_name, num_vowels, num_consonants, and so on
HTML: domain_occurance, depth_DOM, ratio_int_ext_resources, num_uniq_file_types, and so on
Prediction Probability
Model $\mathcal{M}$
(b)

Figure 3. The pipeline for building a supervised phishing detection model. Model $\mathcal{M}$ (a) is trained offline using a labeled data set and (b) used for inference during online operation.

they rely on crawlers that must analyze webpage content beforehand. Such crawlers, in turn, are susceptible to evasion attacks[5] that make phishing webpages appear benign whenever they are accessed by a crawler. Blacklist-based approaches also inherently struggle when attackers use redirection links as lures, as these benign-looking links differ from the URL of the corresponding phishing webpage. The aforementioned limitations of large-scale antiphishing systems have motivated researchers to explore learning algorithms to develop phishing detection solutions capable of running in multiple different contexts and that can keep up with the evasion efforts of attackers.

Supervised Machine Learning: A Brief Overview

A supervised machine learning (ML) algorithm takes a large labeled data set as input to train a classification model that subsequently classifies an input data point into a given number of classes. Figure 3 presents an ML pipeline for developing supervised models that detect phishing attacks. In a phishing webpage detection problem, there are only two classes—benign and phishing—and hence the trained models are binary classifiers. Each data point (e.g., a URL) in the input data set is accompanied by a ground truth label of *benign* or *phishing* to help the model learn the discriminative characteristics of benign and phishing data. Let $\mathcal{M}$ denote such a trained model. In operation (also called the *inference phase*), this trained model $\mathcal{M}$ is given a webpage URL, and it computes the probability p that the webpage is a phishing webpage. Given a configuration threshold θ, the input (the URL) is classified as phishing if the prediction probability is greater than or equal to this threshold, i.e., if $p \geq \theta$, and benign otherwise.

Performance Metrics

The accuracy of a model is dependent on a few factors, some of the important ones being the model algorithm, quality and quantity of data used for training, and value of the classification threshold used in operation. A phishing webpage correctly classified is considered a true positive (TP), and a benign webpage correctly classified is a true negative (TN). Conversely, a phishing webpage wrongly classified (as a benign page) is a false negative (FN), and a benign webpage misclassified as phishing is a false positive (FP). The most important metrics for evaluating phishing detection models are 1) the true positive rate, also called the *recall*, and 2) the false positive rate (FPR). The recall of a model is defined as TP/TP+FN, and the FPR is defined as TP/TP+FN.

The performance of a classification model is a tradeoff between the recall and FPR; e.g., a 100% recall can be achieved by classifying every webpage as phishing (i.e., when $\theta = 0$), but that would also result in the highest FPR for that model. A good evaluation strategy is to measure the model's recall at varying values of the FPR. An FPR of 10^{-1} means, on average, one in 10 benign webpages visited by a user is misclassified as phishing. If users are accordingly warned or blocked, then an FPR of 10^{-1} is clearly unacceptable due to the high number of disruptions caused to legitimate browsing. Therefore, in practical deployments, phishing detection solutions need to evaluate the recall at FPR values of 10^{-3} and even lower.[6,7]

Learning Algorithms for Phishing Detection

In the following, we discuss different ML techniques for phishing detection. We distinguish solutions based on the input they process (for training and prediction)—URLs, HTML content, and webpage

Table 1. The taxonomy of phishing detection solutions based on learning models.

Number	Data	Model	Advantage	Disadvantage
1	URL strings	ML-based model with features engineered from URL strings	Fast No network latency involved	Limited information in URLs Manual engineering of features
2	URL strings	DL-based, with URL strings provided as input to the neural network	Fast No network latency involved No feature engineering required	Limited information in URLs
3	Webpage contents	ML-based, with features engineered from HTML body (in addition to the URL) of a webpage	Contents provide rich information for models to learn and differentiate between benign and phishing webpages	Network latency in obtaining the page contents and expensive feature extraction for real-time detection Susceptible to evasion techniques
4	Webpage screenshots	DL models to compare webpage screenshots and visual invariants, such as logos	Not dependent on large labeled (benign and phishing) data sets. Only small reference list of top targeted websites needs to be maintained	Cost of network latency to fully load a webpage and take a screenshot Prone to adversarial ML attacks

DL: deep learning.

screenshots—as this leads to different deployment use cases. Table 1 provides a taxonomy of such detection techniques.

Phishing URL Classification

A phishing URL classifier trains a classification model on a data set of benign and phishing URL strings along with the ground truth labels (i.e., in a supervised way). Researchers often use top-ranking websites from Alexa (https://www.alexa.com/topsites) and Tranco (https://tranco-list.eu) to build a data set of benign URLs, whereas the set of phishing URLs is usually obtained from PhishTank (https://phishtank.org/) and OpenPhish (https://openphish.com/). Once a model is trained, in operation, it is given a URL obtained from a user (e.g., from a user's browser or email or from a middlebox, such as a web proxy) to make a prediction about whether it links to a phishing page or not. Note that during both the training and prediction phases, these models process URLs only to extract useful information called *features*; they do not visit the URLs. Thus, there is no additional cost for webpage visits, network latency, and page load times for a URL-based ML solution. A URL can be readily processed and a prediction made before the corresponding webpage is visited.

ML-based phishing URL classifier. Early classification solutions engineered features from URLs to train ML models (Table 1, row 1). Features essentially try to capture the lexical properties of URLs; they include the length of a URL, length of the domain part of a URL, length of the path in a URL, number of permitted nonalphanumeric characters (such as dots and hyphens) encoded in a URL, number of vowels and consonants, and so on. Besides, more complicated features are engineered by learning a vocabulary of words (or tokens) and representing the presence of each word in a URL (separately for different components of the URL) as a long binary vector.[8] This results in long feature vectors, which affect the performance of some of the conventional ML models, such as support vector machines. There are also external features, including Internet Protocol (IP) and domain reputation, domain registration information from the WHOIS database, and so on, that are useful for modeling. However, extraction of these external features results in additional network latency or requires maintenance of up-to-date, large databases.

A drawback here is that one has to manually define and engineer discriminative features, and the large body of research works in this direction is evidence that defining all relevant high-quality features exhaustively is a challenging task. Since the phishing attack ecosystem keeps evolving, the list of features needs to be continuously updated. It is likely that, at any point in time, the list of features used in a model is incomplete. Thus, more recent research works apply deep learning (DL) models to detect phishing URLs.

DL-based phishing URL classifier. The convolutional neural network (CNN) is a state-of-the-art DL model that has been widely used for visual analysis while also being applied for text classification. For building a DL-based phishing URL classifier (Table 1, row 2), we first transform the input to a vector of numbers, which is referred to as *embedding*. Each character in a URL is represented by a vector of fixed dimension, and thus each URL is represented as a matrix, with each row representing a character. Since the matrix dimension is also fixed, the number of characters processed in a URL is predetermined; i.e., the length of processed URLs is limited. A CNN model essentially performs convolution operations on the input at its different nodes to learn differentiating patterns among phishing and benign URLs in a labeled data set; this model has been shown to be effective in detecting phishing URLs.[9]

However, character-level models might not be able to capture relationships among characters and, more importantly, words that are far apart in a URL. This is where a sequence model, often used in language modeling, is useful. The long short-term memory (LSTM) architecture, a widely used sequence model, learns dependencies among characters in a long sequence (or string, as in a URL). The basic unit of LSTM has multiple gates that allow the model to learn (and forget) information from arbitrary points in the sequence. Lee et al., section 6.1,[7] evaluated the performance of a CNN, LSTM, and a combined CNN–LSTM architecture on a large data set of URLs obtained from enterprise customers. At a low FPR of 10^{-3}, the combined CNN–LSTM model performed significantly better (achieving a recall of ≈76%) than the independent models (that achieved a ≈58% recall) in detecting phishing URLs. In this combined model, the output of the CNN is fed to LSTM, and that of LSTM is used for prediction. The architecture consists of an embedding layer followed by 256 1D convolution filters, a pooling layer, 512 LSTM units followed by a dropout layer, a dense layer, and, finally, a single unit for classification (using a sigmoid activation function).

Limitations. The information available to train a classification model is limited to what is available in a URL. An attacker today has many options to decide on a domain name, the prominent part of a URL, and the rest of the URL (following the domain name), i.e., the path of a file under the website, is completely under the attacker's control. This makes it easy for an attacker to evade a URL-based detection model. Besides, shortened URLs present much less information for a model to make a good prediction. It is therefore not surprising that URLNet,[9] a URL-based DL model, incurs large number of FPs in a phishing discovery study conducted in the Internet (Lin et al., section 6[6]). One way to overcome this limitation is to develop models based on webpage contents and screenshots. We discuss them in the following.

Phishing Content Classification

The contents of webpages offer a wealth of information that can be exploited to detect phishing sites (Table 1, row 3). Since the goal of an attacker is to deceive users into providing their sensitive information, phishing webpages often have HTML forms and other discriminating characteristics. Therefore, the features that capture the presence of HTML forms, `<input>` tags, and sensitive keywords that prompt users to provide usernames, passwords, credit card numbers, and so on are useful in building an HTML-based phishing classifier.[10] There is also a number of other features useful for modeling; they include the length of the HTML `<form>`, `<style>`, `<script>`, and comments; number of words in text and titles; number of images and inline frames; presence of hidden content; ratio of domain names in the HTML body; and ratio of the number of external links to the number of links under the same domain. In addition, features from URLs are used along with features engineered from webpage contents.

Once the features are defined, a supervised model, e.g., a random forest, is trained using a set of labeled phishing and benign webpages (e.g., from Alexa top-ranking websites and PhishTank); note that these webpages typically have to be crawled to create such a data set. Among the set of URL and HTML features considered, experiments show that some of the important features useful for classification are "the number of times the domain name (of the URL) appears in the HTML contents," "the number of unique subdomains (of the main domain in the URL) present in the HTML contents," and "the number of unique directory paths for all files referenced in the HTML contents."[11] Although the top features might differ among studies depending on the time and source of data (in addition to other aspects, such as the model used), the HTML features tend to be more useful than URL features in the classification of phishing pages has been observed in another recent study.[12]

Limitations. In comparison to a URL-based model, although the content-based model is more accurate, the process of feature extraction from webpage contents is expensive (Xiang et al., section 9.2[10]). The cost of feature extraction during model training is tolerable (since it is an offline process), but during inference, this cost affects the latency experienced by a user. Before allowing a user to access a webpage, the model has to extract features from the webpage and make an inference. HTML-based models are also susceptible to evasion attacks. For example, the three important features mentioned in the preceding can be manipulated by an attacker to evade a phishing classifier. As demonstrated in Lee et al.,[11] one way to counter such an attack (while maintaining detection capability) is to train the classifier with added noise in such a way that the feature importance distribution becomes uniform instead of skewed to a small set of features.

Besides carefully designing HTML pages and contents, attackers also employ cloaking techniques.[2] For example, the use of JavaScript for dynamic content rendering (often requiring user interaction) makes it challenging for automated phishing detection solutions to extract malicious web contents. A recent phishing campaign targeting Microsoft Office 365 users employed multiple JavaScript codes hosted in another server to stack together a page mimicking the Office 365 login interface by using Microsoft logos.[1] The page prompted users to provide their credentials to log in to their accounts. To counter such evasion techniques, an interesting approach is to directly analyze screenshots of webpages.

“Another idea is to infer whether a given webpage uses logos of websites in the reference list of well-known companies and brands while having a different domain.”

Phishing Detection Based on Webpage Screenshots

Phishing websites are effective in deceiving users, due to the visual similarity of a phishing webpage to a well-known website that is being impersonated. A phishing page targeting, say, Facebook users might try to imitate the look and style of the Facebook website (see Figure 2). To achieve visual similarities, attackers use the logos of the target brand/company (e.g., Facebook's); structure, style, and color of the legitimate website (`www.facebook.com`); contents of the target website; and so on while deploying a phishing webpage. This understanding opens up a different direction of research for detecting phishing pages, that which looks for webpages impersonating well-known legitimate brands but having domain names different from targeted brands. For example, if a webpage looks visually similar to that of Facebook's (`www.facebook.com`) but has a different domain (say, `my.famous.co/9z8dccn408`), then the webpage is very likely a phishing page.

DL for screenshot-based phishing detection. The tremendous success of DL in the domain of computer vision has spurred new solutions in screenshot-based phishing detection (Table 1, row 4). Siamese networks and their enhancements learn the similarity of vector representations of given images. This is achieved by training a model in a supervised way, with positive (similar) images and negative (dissimilar) images, via a loss function that decreases the dissimilarity of positive images (since they are supposed to look similar) and increases the dissimilarity of negative images. Put differently, the model is trained to differentiate similar and dissimilar images. In VisualPhishNet,[13] screenshots from a reference list of protected websites are used to train a Siamese model; subsequently, in operation, the trained model estimates the similarity between a given (potentially) suspicious webpage screenshot and all the screenshots in the reference list. A webpage that has high similarity with any website in the reference list while having a different domain is considered a phishing page.

Another idea is to infer whether a given webpage uses logos of websites in the reference list of well-known companies and brands while having a different domain. The problem is broken into two subproblems: one to detect and extract logos from a given webpage and the other to match the extracted logo(s) with those in the reference list of websites. A recent proposal, Phishpedia,[6] uses an object detection model for the first task and a Siamese network for the second task. Another work went further, modeling additional salient user interface features, such as input forms, buttons, text labels, and block structures, on screenshots.[14] Both VisualPhishNet and Phishpedia demonstrate high accuracy in detecting phishing webpages, and they achieve this by analyzing similarities with known images (screenshots and logos on screenshots), which, in turn, are learned from a reference list of websites. Thus, the main dependency is on only a small list of websites that form the protected reference list.

A reference list of the top 100 commonly targeted legitimate websites covers around 95% of all phishing attacks.[6,14] Hence, VisualPhishNet and Phishpedia use a reference list of only a few hundred websites to achieve a high detection rate. A general pipeline for such screenshot-based phishing detection solutions is depicted in Figure 4. Differing from URL- and content-based approaches, screenshot-based solutions that use reference lists typically do not require a large labeled data set of both benign and phishing pages for training models.

Limitations. The dependence on webpage screenshots in the inference phase is a hurdle for the deployment of screenshot-based phishing detection solutions. To take a screenshot, a webpage has to be first fully loaded on a browser. This end-to-end process can take around 2 s, which is expensive for real-time prediction. Finally, object detection models are prone to adversarial attacks, e.g., gradient-based attacks, such as DeepFool. Phishing attackers may use logos that look visually similar to those of protected brands but that might defeat the underlying object recognition model used in current phishing solutions. Countering adversarial attacks is an active area of research, and simple techniques, including changing the activation to a step ReLU (rectified linear unit), are effective against some well-known attacks.[6] Practitioners need to be aware of emerging

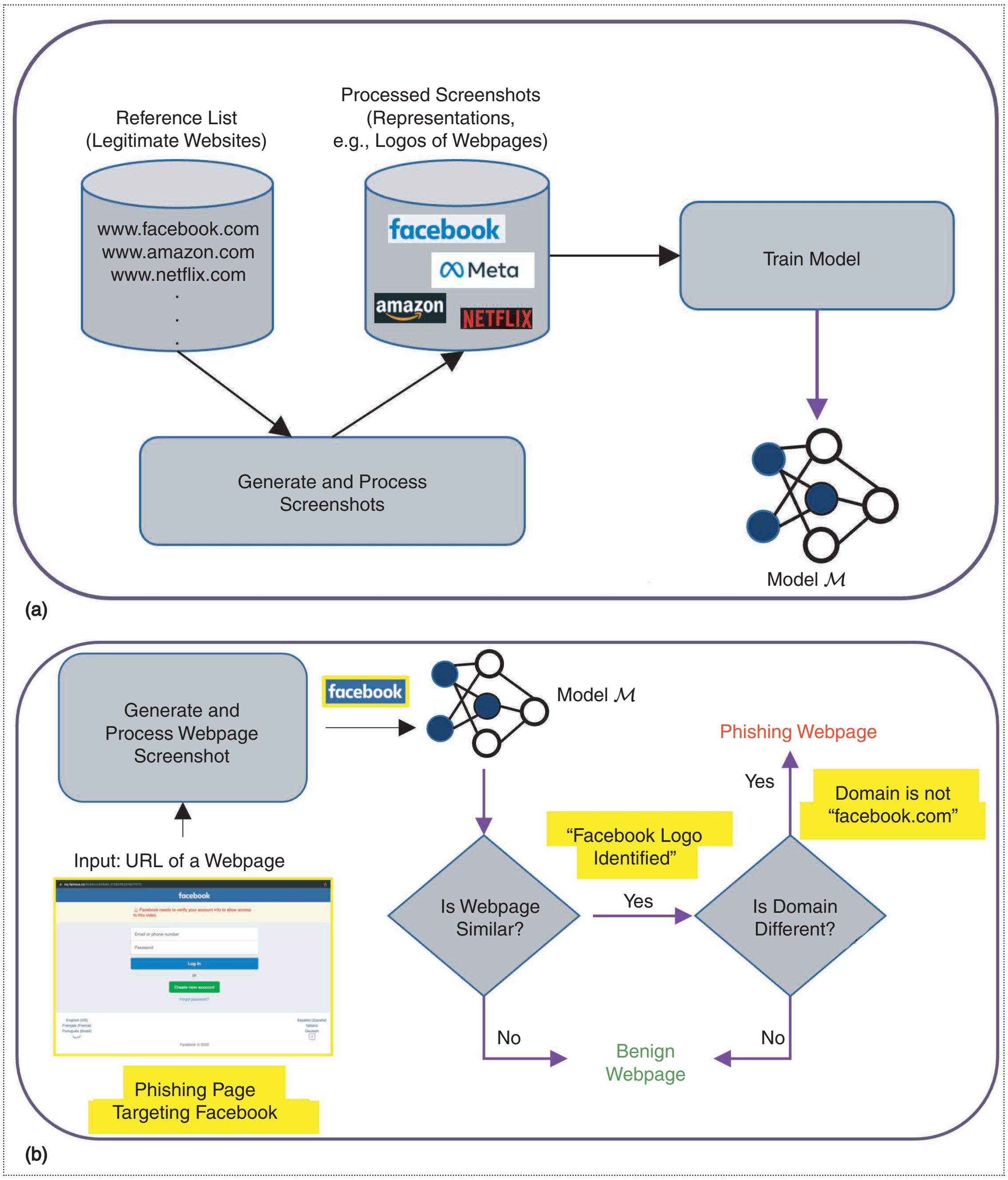

Figure 4. The pipeline for screenshot-based phishing detection. (a) The training phase: model $\mathcal{M}$ is trained using data generated from a reference list of legitimate websites. (b) The inference phase: trained model $\mathcal{M}$ is used in operation. Highlights in yellow illustrate the detection of an example phishing page (given in Figure 1).

adversarial attacks to continuously enhance deployed models for effective mitigation.

Comparison of Different Phishing Solutions

Table 1 presents the key advantages and disadvantages of the different phishing detection solutions. While each approach has been shown to have high accuracy on the data set considered for the corresponding study (e.g., URLNet,[9] CANTINA+,[10] and Phishpedia[6]), few works have compared across the different methods. Yet, intuitively, it is clear that content-based solutions outperform URL-based ones simply because the latter consists of only a small subset of features (or information) used by the former. Evaluations in Li et al.[12] show that HTML features contribute significantly to the high detection accuracy (recall) of ≈97% at an FPR of ≈0.016 achieved by the model (a similar observation is made in Lee et al.[11]). These studies were performed on an offline data set, as is often the case. But that also means that the biases, errors, obsolete links due to short life spans of phishing webpages, and so on, hardly affect the performance of the models being studied, as they are trained and tested on (different partitions of) the same data set. In practice, such supervised models need to be regularly retrained to be effective.

A recent work conducted an experiment for one month on the Internet to evaluate the capabilities of different phishing detectors (Lin et al., section 6[6]). Of the top 1,000 predictions, a content-based (using both URL and HTML features) phishing detector had only nine TPs; whereas, a screenshot-based detector had more than 900 TPs. While the study limited validation to the top 1,000 predictions, the performance gap between the two models was also exacerbated by the fact that the content-based model was trained only once, in an offline manner, using a labeled data set. Besides, since phishing attacks evolve rapidly, the performance of such models degrades quickly with time. On the other hand, the key advantage of the screenshot-based model Phishpedia[6] is that it learns logos of brands from a small reference list (<200) and, importantly, does not depend on a labeled data set; this explains its high detection capability (also confirmed in another work, Liu et al.[14]). Given the preceding studies, note that the URL-based and content-based models need to be retrained frequently using fresh data sets to keep them up to date. Google trains its content-based classifier daily,[15] and generally in the industry, URL- and content-based models are retrained at least once in a week, using millions of new URLs so as to mitigate the problem of performance degradation of models.

“Practitioners need to be aware of emerging adversarial attacks to continuously enhance deployed models for effective mitigation.”

Deploying Phishing Detection Models

Using Multiple Solutions to Complement One Another

While URL-based phishing detection is the fastest, content- and screenshot-based solutions learn from more informative data (page contents and screenshots, respectively). But the latter two incur latency, as a webpage has to be fully loaded before the models can be applied. A screenshot-based model has to additionally capture a screenshot and subsequently perform operations (e.g., logo detection and identification in Phishpedia[6]) on it before making a prediction. One technique to take advantage of these models is to build a pipeline of all three (working on three different data sources).

Consider a phishing detection pipeline wherein URLs are first fed to a URL-based solution (e.g., URLNet[9]) in real time. In the next stage, phishing URLs with high prediction probability (say, $\geq \theta$) are streamed to a content-based model (e.g., CANTINA+[10]). Thus, the number of URLs for which entire pages will be fetched is controlled using the threshold θ. Since a content-based detector has many more informative features, it can effectively reduce FPs due to a URL-based model. Yet, as a content-based model is weak when dealing with dynamic contents, a screenshot-based solution can come next in the pipeline. For pages for which a content-based solution is weak (e.g., rules can detect the presence of scripts in HTML code), the contents can be rendered on a browser, and a screenshot can be taken to stream to a screenshot-based solution (e.g., Phishpedia[6]). Such a pipeline leverages the high detection capability (at a low FPR) of a screenshot-based solution while still putting it to use only for the most difficult pages to classify. The throughput of this pipeline can be controlled by deciding what fraction of the URLs/pages should be streamed to the next stage.

Finally, in practice, security solutions are often multilayered. To minimize FPs and increase the detection rate, classification models are complemented with lists of known benign websites, scoring based on different threat intelligence sources (where we see increasing collaborations within the cybersecurity industry), and the reputation of IP addresses, domains, and hosting services. Next, we discuss specific deployment options.

Integration With Secure Email Gateways

One of the most common ways of delivering phishing URLs to users is via emails. Enterprises today deploy secure email gateways that analyze the emails being delivered to their employees to detect, warn about, and block spam, malicious attachments, and phishing emails. Since email is an asynchronous application that tolerates latency, phishing detection solutions based on URLs, HTML contents, and screenshots, are all good candidates for deployment. Besides, two or more models (along with rules and scoring systems) can be deployed as a pipeline.

In addition to URLs, emails contain other important components that are indicative of phishing attacks: headers and subjects, textual contents, and attachments. These surfaces are exploited by attackers for different purposes. Automatically generating and sending phishing emails in large numbers leaves some noticeable attack imprints on email headers and subjects. But if an attacker's goal is to lure a particular user into replying with confidential information, then the email body will be exploited with specific contents to persuade the user. A solution that extracts information from multiple components of emails has the potential to detect different kinds of phishing attacks[7] and is naturally aligned to be integrated with secure email gateways for detecting phishing emails.

Integration With Browsers

For browser users, phishing inference should be made before a webpage is loaded. Users typically expect webpages to load within a few hundreds of milliseconds. Thus, screenshot-based phishing detection is not a good candidate since screenshots require webpages to be fully loaded, and this usually takes a couple of seconds due to network latency, multiple components of pages being loaded from different servers, JavaScript execution, displaying complex objects, and so on. Between URL- and content-based solutions, the former is better suited for integration with browsers, as they process only URLs to make inferences. Content-based models can still be put to use: Google deploys a content-based model to classify URLs that it receives via other sources (such as Gmail) to build a blacklist,[15] and this blacklist is integrated with Google's Chrome browser.

Independent Analyzers

All phishing solutions can run as independent analyzers processing URLs delivered to them. VirusTotal (www.virustotal.com) runs tens of engines contributed by different vendors to analyze user-submitted URLs (which is rate limited without a paid service subscription). Integrating with VirusTotal, a phishing detection solution would get new and potentially suspicious URLs submitted by users around the world. The latency requirements of such engines are not as stringent as those for browsers since users are typically prepared to wait for a few seconds for the results. Thus, URL- and content-based phishing detectors are good candidates for integration with systems such as VirusTotal. Alternately, as demonstrated in Lin et al.[6] and Liu et al.,[14] solutions can integrate with CertStream (https://certstream.calidog.io/), a service that provides a feed of URLs with newly issued certificates in real time.

We discussed different models for detecting phishing attacks, based on the input considered: URLs, HTML contents, website screenshots, and emails. We also presented practical deployment scenarios for the models. Based on the current state of the research as well as experience in the industry, we conclude with the following takeaways:

1. By automatically learning from large-scale data, learning models form an important set of tools to counter phishing attacks.
2. Phishing detection models have to achieve good recall at a very low FPR for usable security; an FPR of 10^{-3} is usually expected for adoption in the industry.
3. Besides being evaluated on an offline data set, a longitudinal study in the wild is useful for ensuring that a proposed model can be generalized.
4. Different approaches (discussed in this article) have different pros and cons; besides, learning models are inherently probabilistic and not perfect. Therefore, a pipeline of multilayered approaches, including rules and scores based on other traditional sources, is required in practice.
5. Phishing detection models are prone to different evasion strategies that are technically feasible; in fact, we have evidence of phishing attacks with smart evasion techniques.[1] We identify two important hurdles for phishing detection solutions: 1) cloaking techniques employed by attackers[2] and 2) adversarial attacks that target computer vision-based DL models. Research into next-generation antiphishing systems should focus on robust approaches for overcoming these challenges. ■

References

1. H. Pacag, "HTML Lego: Hidden phishing at free JavaScript site." Trustwave. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/html-lego-hidden-phishing-at-free-javascript-site/
2. P. Zhang *et al.*, "CrawlPhish: Large-scale analysis of client-side cloaking techniques in phishing," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2021, pp. 1109–1124, doi: 10.1109/SP40001.2021.00021.
3. A. Oest *et al.*, "Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 1–17.
4. M. Bitaab *et al.*, "Scam pandemic: How attackers exploit public fear through phishing," in *Proc. APWG Symp. Electron. Crime Res. (eCrime)*, 2020, pp. 1–10, doi: 10.1109/eCrime51433.2020.9493260.
5. A. Oest *et al.*, "PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 379–396.
6. Y. Lin *et al.*, "Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages," in *Proc. 30th USENIX Security Symp.*, 2021, pp. 3793–3810.
7. J. Lee, F. Tang, P. Ye, F. Abbasi, P. Hay, and D. M. Divakaran, "D-Fence: A flexible, efficient, and comprehensive phishing email detection system," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, 2021, pp. 578–597, doi: 10.1109/EuroSP51992.2021.00045.
8. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious URLs: An application of large-scale online learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 681–688, doi: 10.1145/1553374.1553462.
9. H. Le, Q. Pham, D. Sahoo, and S. C. Hoi, "URLNet: Learning a URL representation with deep learning for malicious URL detection," 2018, *arXiv: 1802.03162*.
10. G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "CANTINA+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Trans. Inf. Syst. Security*, vol. 14, no. 2, pp. 1–28, Sep. 2011, doi: 10.1145/2019599.2019606.
11. J. Lee, P. Ye, R. Liu, D. M. Divakaran, and C. M. Choon, "Building robust phishing detection system: n empirical analysis," in *Proc. Workshop Meas., Attacks, Defenses Web (MADWeb)*, Feb. 2020, pp. 1–12, doi: 10.14722/madweb.2020.23007.
12. Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, "A stacking model using URL and HTML features for phishing webpage detection," *Future Gener. Comput. Syst.*, vol. 94, pp. 27–39, May 2019, doi: 10.1016/j.future.2018.11.004.
13. S. Abdelnabi, K. Krombholz, and M. Fritz, "VisualPhishNet: Zero-day phishing website detection by visual similarity," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2020, pp. 1681–1698, doi: 10.1145/3372297.3417233.
14. R. Liu, Y. Lin, X. Yang, S. H. Ng, D. M. Divakaran, and J. S. Dong, "Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach," in *Proc. USENIX Security Symp.*, 2022.
15. C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2010.

Dinil Mon Divakaran is a senior security researcher at Trustwave, heading R&D within the office of the chief technology officer. He is also an adjunct assistant professor in the School of Computing, National University of Singapore (NUS), 117417 Singapore. He has more than a decade of experience leading R&D in both industry and academia, working on topics of interest that include phishing detection, large-scale security log analysis, Domain Name System security and privacy, Internet of Things network security, and programmable in-network security. His research interests include network and system security, artificial intelligence for security, protocol analysis, and programmable data planes. Divakaran received a Ph.D. at École Normale Supérieure de Lyon, France, in the joint lab of the National Institute for Research in Digital Science and Technology and Bell Labs. Contact him at dini.divakaran@trustwave.com.

Adam Oest is a senior cybersecurity researcher at PayPal, Inc., Scottsdale, Arizona, 85258, USA, where he leads a team focused on identification and mitigation of sophisticated fraud and strengthening the security of the broader web ecosystem. He is passionate about Internet security. Oest received a Ph.D. in computer science from Arizona State University, Tempe. His recent work on antiphishing has won distinguished paper awards from *USENIX Security* and *IEEE Security & Privacy*, and he was the lead author of the 2020 Internet defense prize-winning paper "Sunrise to Sunset: Analyzing the End-to-End Life Cycle and Effectiveness of Phishing Attacks at Scale," which presents a framework for the proactive detection and end-to-end measurement of phishing attacks. He is also a principal investigator for the PhishFarm Block List Latency Monitoring Program with the Anti-Phishing Working Group. Contact him at aoest@paypal.com.